НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено
Завідувач кафедри
_____ *С.Г.Стіренко*
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломний проект

освітньо-кваліфікаційного рівня « бакалавр »

з напряму підготовки(спеціальності) 6.050103 «Програмна інженерія»

на тему «Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS»

Виконав: студент *4* курсу, групи *ІП-53*
Іванець Олександр Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *старший викладач Алещенко О. В.* _____
(прізвище, ім’я, по батькові) (підпис)

Консультант *нормоконтроль* *Симоненко В. П.* _____
(назва розділу) (прізвище, ім’я, по батькові) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО”

Факультет Інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Обчислювальної техніки
(повна назва кафедри)

Освітньо-кваліфікаційний рівень **бакалавр**

Напрям підготовки **6.050103 « Програмна інженерія »**

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(ініціали, прізвище) (підпис)

“___”_____2019р.

ЗАВДАННЯ

на бакалаврську дипломну роботу студента

Іванця Олександра Андрійовича
(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS
керівник проекту (роботи) Алещенко Олексій Вадимович, старший викладач ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “23” 04 2019 року №1180-С
2. Термін здачі студентом закінченого проекту (роботи) 20 червня 2019р.
3. Вихідні дані до проекту (роботи) технічна документація, теоретичні дані та функціонал для виконання замовлень в системі придбання товарів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) Опис та аналіз предметної області, аналіз аналогів у даній області, дослідження основних вимог до ПЗ, конструювання архітектури системи та створення компонентів системи придбання товарів з використанням місцезнаходження користувача, інструкція користувача
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, узагальнена схема процесу придбання товару, діаграма класів

6. Консультанта проекту (робота), з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	Затвердження теми роботи	10.12.2018 - 15.12.2018	
2.	Вивчення та аналіз завдання	15.12.2018 - 15.03.2019	
3.	Розробка архітектури та загальної структури системи	15.03.2019 - 25.03.2019	
4.	Розробка структур окремих підсистем	25.03.2019 - 05.04.2019	
5.	Програмна реалізація системи	05.04.2019 - 15.04.2019	
6.	Оформлення пояснювальної записки	15.04.2019 - 25.05.2019	
8.	Передзахист	29.05.2019	
9.	Захист	20.06.2019	

Студент-дипломник _____
(підпис)

Керівник роботи _____

(підпис)

	Формат	Позначення	Найменування	Кільк сторінок	№ екземпляру	Примітка
1	А4		Завдання на дипломний проект	2		
2	А4	ІАЛЦ.467200.001 ВП	Відомість на дипломний проект	1		
3	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4		
4	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	60		
5	А4	ІАЛЦ.467200.004 Д1	Діаграма класів	1		
6	А4	ІАЛЦ.467200.005 Д2	Діаграма компонентів	1		
7	А4	ІАЛЦ.467200.006 Д3	Діаграма послідовності	1		
8	А4	ІАЛЦ.467200.007 Д4	Лістинг програми	24		
9	А4	ІАЛЦ.467200.008 Д5	Скриншоти програми	5		
10	А4	ІАЛЦ.467200.009 Д6	Алгоритм програми	1		

					ІАЛЦ.467200.001 ВП
Изм	***Лист***	***№ докум.***	***Подп***	***Дата***	
Розробив		Іванець О.А.			Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS
Перевір.		Алещенко О.В.			
Н. контр.		Сімоненко В.П			НТУУ «КПІ» ФІОТ ІП-53
Затв.		Стіренко С. Г			

АНОТАЦІЯ

При розробці даної дипломної роботи будо розроблено програмний продукт для операційної системи iOS, головна мета якого покращення комунікації власників бізнесу в спортивному сегменті з їхніми користувачами, за допомогою роботи програмних та апаратних можливостей мобільних девайсів, таких як місцезнаходження користувача, оновлення в поточному часі даних з віддаленої бази даних на платфорті Firebase, виділення товарів з найвищою ступенню продажі і рекламних оголошень. Програмний продукт забезпечує користувача таким функціоналом, як: можливість переглянути товари категорії, магазину, отримувати повідомлення про рекламні оголошення при знаходженні користувача в одному кілометрі від магазину, зробити замовлення та оплатити його через інтегровану платіжну систему Fondy.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ ПРОЦЕСІВ, ЕЛЕКТРОННА КОМЕРЦІЯ, ГЕОЛОКАЦІЯ.

ABSTRACT

During development of this diploma was created the program product for operation system iOS, the main goal of which increasing of communication between business owners in sport segment and their customers via software and hardware abilities of mobile devices, such as checking user location, realtime updates of local data with remote database within Firebase platform,

stuffs aggregating with the highest level of sales and advertisements. Program product provides user with such features: ability to look through stuffs in specific category, store, get notifications about current offers while being in one kilometer from specific store, make order and pay for it with integrated payment system Fondy.

KEYWORDS: AUTOMATION, ECOMMERCE, GEOLOCATION.

Пояснювальна записка
до дипломного проекту

на тему: «Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS»

Київ – 2019 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання спрямоване на розробку мобільного застосунку «Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS». Область застосування: платформа для покращення ефективності пошуку необхідних товарів у сфері спорту за допомогою використання місцезнаходження користувача, використання платформи Firebase в якості бази даних та для обробки аутентифікаціїї користувачів, виокремлення товарів з більшою кількістю замовлень.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Завдання на виконання дипломного проекту освітньо-кваліфікаційного рівня «бакалавр», затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного мобільного додатку є збільшення задоволення та ефективності використання систем покупок товарів. Призначення додатку: покращення комунікації бізнесів торгівлі та користувача.

4. ДЖЕРЕЛА РОЗРОБКИ

На даний час існують багато рішень, такі як Amazon, Rozetka та інші, але ні один з сервісів не може запронувати можливість об’єднання багатьох інших магазинів в одному застосунку задля того, щоб відправити користувачу інформацію про товар в залежності від його геопозиції.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Захищеність даних.
- Забезпечення користувача нотифікаціями з інформацією про гарячі пропозицію при проходженні в радіусі 1 кілометра від конкретного магазину.
- Можливість оформити замовлення в різних магазинах.
- Можливість оплати замовлення прямо в застосунку.
- Використання бази даних Firebase для можливості її використання на девайсах з ОС iOS та Android.

5.2. Вимоги до програмного забезпечення

- Операційна система iOS 10.0 і вище

5.3. Вимоги до апаратного забезпечення

- Мінімальні технічні вимоги:
    - Частота процессора: 1.3 ГГц.
    - Мінімальний об’єм оперативної пам’яті: 1ГБ.
    - Вільне місце не менш ніж 80МБ.
- Бажані технічні вимоги:
    - Частота процессора: 1.8 ГГц.
    - Об’єм оперативної пам’яті: 1.5ГБ.
    - Вільне місце не менш ніж 120МБ.

6. ЕТАПИ РОЗРОБКИ

Етап	Дата
1. Вивчення літератури за тематикою проекту	24.11.2018
2. Складання і узгодження технічного завдання	25.12.2018
3. Аналіз області застосування та особливостей системи	15.01.2019
4. Розробка загальної архітектури системи	15.03.2019
5. Реалізація програмного забезпечення	05.04.2019
6. Тестування програмного забезпечення	13.04.2019
7. Оформлення документації дипломної роботи	15.04.2019

Технічне завдання
до дипломного проекту

на тему: «Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS»

Київ – 2019 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CMS – програма для створення та підтримання веб-сайтів.

SaaS – модель роботи сервісу, коли користувачеві надають вже готовий функціонал без потреби в розгортанні його користувачем

IP – інтернет-протокол

API – інтерфейс прикладного програмування

DB – база даних

UI – інтерфейс користувача

ВСТУП

Кожного року людство стає все більш й більш диджиталізованим, технології стають більш доступними й легкими для використання для великих мас населення. Одним з чинників глобальної диджиталізації стає те, що пристрої стають все дешевшими, потужнішими й насамперед, компактнішими.

З рістом кількості мобільних девайсів, вдосконаленням засобів та технологій розробки почала збільшуватися кількість мобільних додатків, вони почали покривати більше аспектів й моментів з повсякденного життя населення, що дозволяє менше часу приділяти рутинним справам і витрачати цей час на інші активності.

Однією з давніших галузей, котру с розвитком технологій почали массово переносити в світ інтернет технологій, являється торгівля. В наш час неможливо уявити собі якійсь конкретний торгівельний бізнес, котрий був би масовим, успішним та не надавав свої послуги через веб сайти й мобільні додатки. За наявності у користувача мобільного пристрою та мобільного додатку, він може дуже швидко перевірити навність речі, котра його цікавить, перевірити ціну, порівняти з аналогами, зробити остаточний вибір, обрати місце доставки та придбати товар, і все це він може зробити на відстані від самого магазину, не витрачаючи свого часу.

Сучасні рішення, присутні на ринку, не мають деякої частини функціоналу, котра могла б покращити досвід їх використання . Декотрі з них супроводжують користувача від початку до покупки товару, але зосереджені на конкретно одному магазині. Інші ж надають можливість отримати інфрормацію про продукт, проте не надають можливості придбати товар, а спонукають користувача переходити на ресурс продавця або магазину, де цей товар є в наявності. Агрегувавши різні магазини в один комлексний сервіс стає

можливим інформувати користувача про знижки чи спеціальні пропозиції в магазинах, які знаходяться не далеко від нього.

Мета створення данної роботи – вдосконалення сервісів, спрямованих на торгівлю товарами, шляхом об’єднанням їх в одну централізовану систему, покращити досвід використання сервісом шляхом нотифікування користувача про спеціальні пропозиції від маркетів, які знаходяться в місці його розташування шляхом розроблення мобільного додатку на платформу iOS.

Завданням роботи є розробка системи, котра агрегуватиме в собі різні магазини з різними категоріями товарів й нотифікувати користувача про спеціальні вигідні пропозиції, які є неподалеку від місця знаходження користувача. Результатом роботи є мобільний застосунок для платформи iOS, котрий можна буде використовувати в перечислених цілях.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні положення

Інтернет торгівля, або ж електронна коммерція – одна з сфер цифрової економіки, суть якої полягає в продажу або купівлі продукції через мережу Інтернет. За останні роки набуває все більших обертів та потрохи витісняє торгівлю в звичайному вигляді через низку переваг, котрі має інтернет торгівля товарами над торгівлею товарами в реальному часі, такими як зручність використання, простота в залишенні відгуків про товар, великий вибір однакових товарів в різних магазинах задля вибору найпривабливішої ціни, можливість швидко знайти схожий товар для порівняння характеристик між декількома товарами.

За допомогою нових технологій, котрі присутні в новітніх девайсах, можна додати новий функціонал, котрий буде додавати користі й зручності в використанні.

1.2 Змістовний опис і аналіз предметної області

1.2.1 Електронна коммерція, її види

В мережі існує велика кількість різноманітних видів електронної комерції. Візуальна частина цих сервісів представлена веб сайтами та/або мобільними додатками, котрі в дечому схожі а в дечому різняться один від одного, саме тому існує декілька видів класифікацій, за допомогою яких їх можна порівняти.

За типом платформи виділяють:

- Власна CMS. Перевагою є унікальність й впізнаваємість дизайну, можливість розширення фунціоналу та адміністрування компонентів, зважаючи на власні потреби. Недоліком такого

підходу є потенційне витрачення фінансів на менеджинг та отримання ліцензії.

- Безкоштовна CMS. Такий підхід дозволить заощадити кошти і надасть можливість навчитися створювати інтернет-магазин своїми силами. Недоліком такого шляху є в те, що початково фунціонал безкоштовних CMS є доволі обмеженим, також неможливістю отримати технічну підтримку, витраченням особистого часу на вдосконалення та адміністрування.

- Базується на SaaS платформі . Замовник отримує вже повністю готовий інтернет-магазин, функціонал якого продуманий до дрібниць. Відвідувачі можуть замовити оперативну підтримку разом з автоматичними оновленнями. Оренда SaaS передбачає внесення щомісячної оплати. Якщо порівнювати цю платформу з власною CMS, то функціонал комерційних сайтів, створених на SaaS, буде в деякій мірі більш обмеженим.

За асортиментом товару:

- Спеціалізовані магазини. Здійснюється продаж тільки певних видів товарів (наприклад, аксесуари для мобільних гаджетів або нижню білизну).
- Віртуальні супермаркети. У таких інтернет-магазинах вам запропонують будь-які товари. Асортимент може включати буквально все від предметів особистої гігієни і домашнього текстилю до великогабаритного устаткування і будівельних матеріалів.

За типом товару:

- Нішеві. Реалізують товари, які користуються високим попитом і мають низьку вартість.
- Брендові. Працюють з фірмовою продукцією, гарантійні зобов'язання на яку, як правило, надаються виробником.
- Змішані. У продажу можна знайти як нішеві, так і брендові товари.

За географією надання послуг:

- Регіональні.
- Територія однієї країни.
- Міжнародні.

За типом роботи:

- Тільки онлайн-бізнес. Компанія здійснює свою діяльність тільки в Інтернеті. В офлайні може мати тільки склад або офіс + склад.
- Інтернет-магазин, який має стаціонарне представництво. Торгівля ведеться і в Інтернеті, і в звичайному магазині або торговому центрі. [1]

Також їх поділяють за тематикою, проте перечисли їх не реально, тому що в такому випадку потрібно просто перечислити абсолютно всі види товарів, котрі можна реалізувати таким чином.

Також їх поділяють за тематикою, проте перечисли їх не реально, тому що в такому випадку потрібно просто перечислити абсолютно всі види товарів, котрі можна реалізувати таким чином.

1.2.2. Місцеположення, методології його знаходження

Велику роль в процесі продажу товару займає місцеположення як магазину, так і знаходження користувача. Маючи інформацію про нинішне місце знаходження людини можна перевірити, чи є сенс їхати в обраний магазин задля покупки обраного товару, чи він може знайти такий самий або схожий за характеристиками товар в місці, яке знаходиться ближче до нього;

чи може сервіс, в котрому користувач зробив замовлення відправити товар в район знаходження користувача; якщо в конкретному магазині, котрий є неподалеку від користувача є локальна акція на товар або групу товарів, нотифікувати його про акцію, в такому разі вигоду отримує як користувач, заощадивши певну суму грошей, так і магазин, який таким чином має можливість швидше реалізувати товар, а отже й прибуток від нього. Для того, щоб визначити локацію людини, можна обрати один або декілька з нижче наведених методологій:

- GPS сервіс
- За IP адресою девайсу
- За допомогою вишок телеоператорів та Wi-Fi мереж
- За допомогою інформації з соціальних мереж
- Розглянемо більш детально кожен з варіантів.

GPS – система електронних виробів, спрямована на знаходження локації та швидкості предмета. Його знаходження розраховується шляхом використання приймача, котрий приймає та опрацьовує сигнали з спутників. В мобільному телефоні місцезнаходження з використанням GPS API є найбільш точним варіантом, тому що інкапсулює в собі не лише сигнали с спутників, а також використовує інформацію з Wi-Fi мереж й інформацію від операторів телекомунікацій. Результатом є можливість отримати інформацію про місцезнаходження в дуже великому спектрі, починаючи від країни знаходження і аж до локації з похибкою в один – два метри.

За IP адресою – доволі швидкий, але не завжди точний спосіб реалізації пошуку локації. Представляє собою звичайну базу, в якій зберігаються безпосередньо IP адрес та інформація про його локацію. Головним недостатком цього методу є те, що цю базу потрібно підтримувати, обновляти інформацію.

За допомогою соціальних мереж – основна маса з присутніх на ринку соціальних мереж надає можливість користувачеві додати свою локацію в

цілях надати список інших користувачів з того ж регіону для простішого пошуку знайомих серед них. Дана схема отримання локації не є ідеальною, адже є великий шанс, що користувач завідомо внесе неправильне місце знаходження, тому задля покращення результату пошуку береться до уваги вказане знаходження друзів з його нетворку, будується модель аналізу цих даних і вже після цього дані використовують. Також цей метод не є досить точним в знаходженні локації, адже таким чином максимум можна виявити назву міста користувача, більш точні данні навряд будуть актуальними.

1.2.3. Генералізований процес покупки через мобільний застосунок

Процес покупки через мобільний застосунок або інтернет магазин можна описати за допомогою діаграми на Рисунку 1.1.

Починається все з того, що продавець створює розміщення товару в себе на сайті або в своєму додатку. Після цього зацікавлений в товарі покупець обирає товар, вводить дані про себе задля визначення місця доставки та верифікації в майбутньому під час отримання доставки. Коли вся необхідна інформація занесена користувач натискає придбати товар, після чого його перенаправляють на сторінку оплати замовлення. Після того, як оплата була верифікована сервісом оплати, користувачеві надається інформація про сформований заказ та про деталі доставки товару. В цей час продавець інформується про успішну оплату товару й відправляє товар покупцеві.

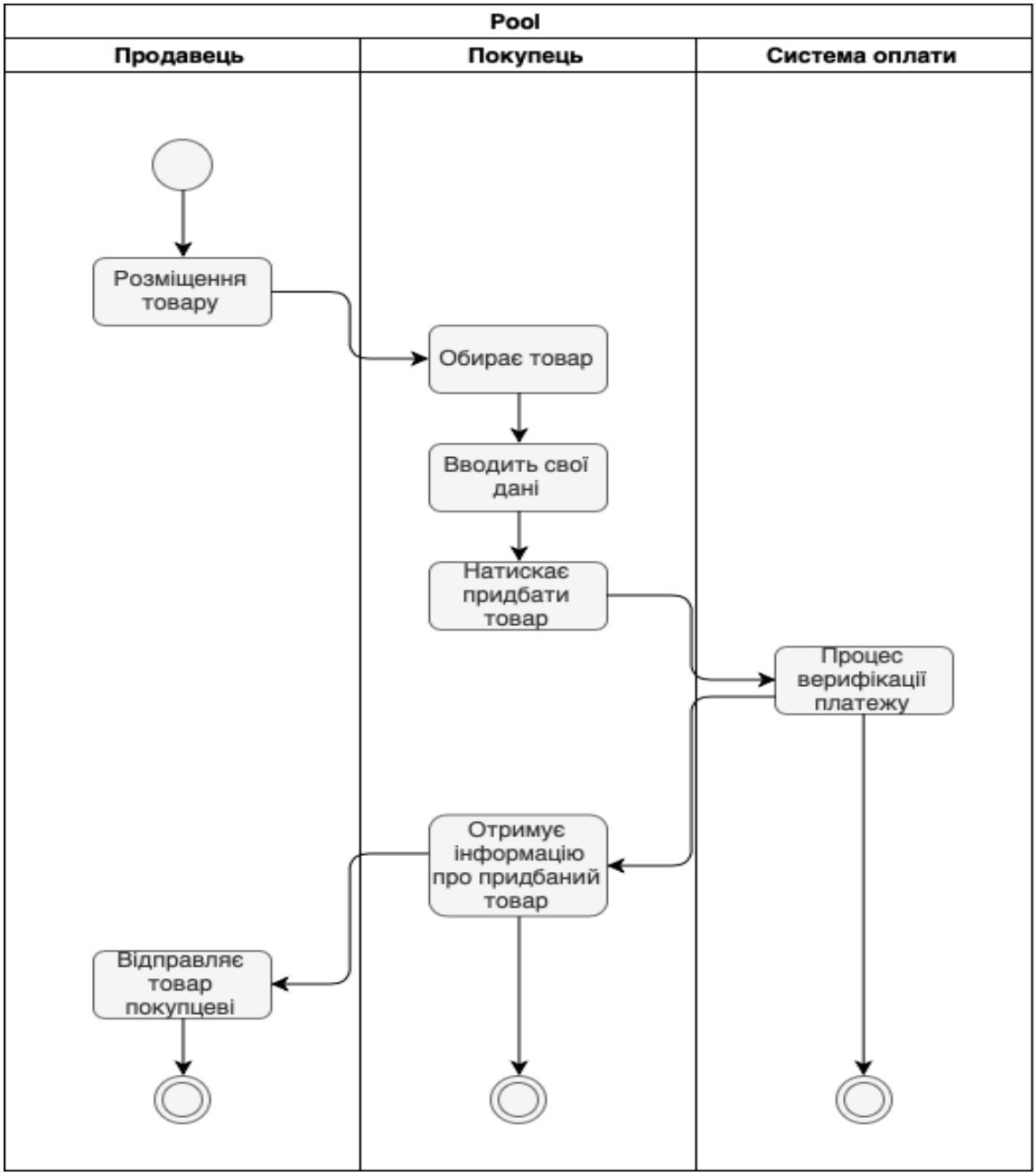


Рисунок 1.1 – Діаграма діяльності учасників процесу покупки

1.3 Аналіз успішних IT-проектів

Так як торгівельна індустрія не нова тема, і конкуренція на ринку доводі сувора, ситуація склалася так, що на ринку залишаються лише ті сервіси, котрі мають певну привілегію над іншими. В основному це нижчі ціни, ніж у конкурентів, велика кількість впроваджених нових технологій та додаткових сервісів, котрі спонукають користувача використовувати саме цей магазин.

Прикладом можуть слугувати різноматні системи накопичувальних знижок, матеріальних бонусів за те, що вони приведуть своїх друзів та знайомих, нові види доставки товарів, розширення зони доставок товарів, проведення акцій з розіграшом товарів або деякої фінансової суми, тощо.

1.3.1 Аналіз відомих програмних продуктів

Найбільш популярні інтернет сервіси з продажу товарів:

Rozetka.ua

Один з найвідоміших сервісів з покупки товарів на території України.

Спочатку сервіс базувався як магазин з продажу електронної техніки, але з ростом аудиторії почали додаватися нові товари, не пов'язані з технікою. На даний момент магазин пропонує доволі широкий спектр різноманітних товарів. З відносно недавного часу сервіс додав можливість розміщення іншим інтернет магазинам в себе на площадці, тим самим даючи користувачам можливість обирати однаковий товар у різних продавців, опираючись на ціну та рейтинг продавця.

Недоліком даного сервісу є недостатньо розвинута система таргетингу особливих пропозицій окремим користувачам в залежності від попередньо придбаних товарів.

Amazon.com

Один з перших інтернет магазинів, котрі були орієнтовані на торгівлю товарами, які користуються масовим попитом. Одна з найбільших компаній, що займається продажом товарів та різних послуг через мережу Інтернет. На даний момент має сайти окремо для кожної з 15 країн. Є одною з найдорожчих компаній в світі.

Починав свій шлях в торгівлі як магазин з продажу книжок, проте з часом також почали з'являтися музична та відео продукція. Являється однією з найтехнологічніших компаній, адже саме вони впровадили новітній спосіб

доставки товарів системою дронів та дирижаблів. Сервіс не перестає експериментувати з асортиментом задля того, щоб відкрити для себе нові види ретейлу.

eBay Inc.

Всесвітньо-популярний інтернет-аукціон. Система даного сервісу дозволяє звичайному користувачеві як і використовувати його з метою покупки товару, так і з метою продажі. Пошук необхідного товару проводиться за допомогою ключового слова або слів, а також може бути урегульованим фільтрами, такими як місце знаходження продавця, товари, котрі можна придбати за фіксованою ціною без попереднього аукціону, тощо. Присутня функція перегляду вже проданих товарів. Система товарів представлена такими видами лотів:

- Аукціон
- Придбання за фіксованою ціною
- Комбінація двох вище перечислених методів, тобто коли товар можна придбати за попередньо виставленою фіксованою ціною, або ж на аукціоні.

Товари можуть бути як новими, так і бувшими у використанні. На даний момент існують філії в шести країнах світу.

1.4 Аналіз вимог до програмного забезпечення

Розроблене програмне забезпечення має мати такий фунціонал:

Аутентифікація користувача за допомогою аккаунту Google або Facebook

- Перегляд списку магазинів, категорів товарів та товарів
- Перегляд свого місцезнаходження та розташування магазинів на мапі
- Перегляд оформлених заказів
- Перегляд особистого профілю

- Можливість додавання товару в кошик та можливість оформити заказ

1.4.1 Розроблення функціональних вимог

Розроблений застосунок має передбачити виконання таких функціональних вимог:

Таблиця 1.1 – Функціональна вимога FR001

Номер	FR001
Назва	Аутентифікація в системі
Опис	Застосунок має надати можливість користувачві аутентифікуватися за допомогою системи Google чи Facebook

Таблиця 1.2 – Функціональна вимога FR002

Номер	FR002
Назва	Перегляд списку магазинів, категорій чи товарів
Опис	Застосунок має надавати користувачеві можливість переглядати списки товарів, їх категорій та магазинів, де вони реалізуються

Таблиця 1.3 – Функціональна вимога FR003

Номер	FR003
Назва	Пошук магазинів чи категорій товарів
Опис	Користувач має бути спроможним знайти магазин чи категорію товарів, які йому цікаві.

Таблиця 1.4 – Функціональна вимога FR004

Номер	FR004
Назва	Додавання товару в кошик
Опис	Користувач може додавати товар в кошик, таким чином його формуючи.

Таблиця 1.5 – Функціональна вимога FR005

Номер	FR005
Назва	Отримання інформації про товар
Опис	Користувач може отримати детальний опис кожного товару, котрий є в наявності.

Таблиця 1.6 – Функціональна вимога FR006

Номер	FR006
Назва	Відображення магазинів на мапі
Опис	Користувач може переглянути представлені магазини на мапі

Таблиця 1.7 – Функціональна вимога FR007

Номер	FR007
Назва	Відображення геолокації користувача
Опис	Користувач може побачити свою геолкоацію на мапі з магазинами

Таблиця 1.8 – Функціональна вимога FR008

Номер	FR008
Назва	Відображення кошика користувача
Опис	Користувач має можливість переглянути товари, котрі він додав в кошик відповідно до кожного з магазинів.

Таблиця 1.9 – Функціональна вимога FR009

Номер	FR009
Назва	Можливість коректувати наявність та кількість кожного товару в кошику
Опис	Користувач може збільшувати, зменшувати чи видаляти кожен з товарів, котрий є в кошику.

Таблиця 1.10 – Функціональна вимога FR010

Номер	FR010
Назва	Придбання товару.
Опис	Користувач може придбати товар або группу товарів з обраного магазину

Таблиця 1.11 – Функціональна вимога FR011

Номер	FR011

Назва	Нотифікація користувача.
Опис	Застосунок може інформувати користувача про спеціальні пропозиції від магазинів неподалеку.

Таблиця 1.12 – Функціональна вимога FR012

Номер	FR012
Назва	Перегляд профілю
Опис	Користувач може переглянути та додати інформацію в власний профіль в додатку

Таблиця 1.13 – Функціональна вимога FR013

Номер	FR013
Назва	Банківські карти користувача.
Опис	Користувач може додавати, видаляти, переглядати та використовувати додані банківські карти під час процесу оплати товару.

1.4.2 Розроблення нефункціональних вимог

Застосунок має підтримувати такі нефункціональні вимоги:

- локалізація інтерфейсу – українська;
- мінімально підтримувана версія iOS – iOS 10.0
- дані з клієнту на сервер та навпаки мають передаватися через протокол https.

ВИСНОВКИ ДО РОЗДІЛУ 1

Торгівля була, є і буде займати велике значення в житті людства. Це є базовий процес для суспільства, завдяки їй ми можемо отримувати те, що нам необхідно, тому є закономірним те, що в цифрову епоху цей процес массово мігрує в мережу Інтернет.

На даний момент існує багато сервісів, котрі займаються електронною коммерцією, проте жодний з них не є ідеальним – ні в одному з них не використовується місцеположення користувача й недалека віддаленість його від конкретного магазину, в котрому нв той момент часу є якась супер-вигідна пропозиція. Також на ринку немає ні одного з сервісів, котрий агрегував би вміст різних магазинів та міг би підтримувати весь процес покупки, а саме верифікація грошової транзакції за конкретне замовлення в конкретному магазині.

Реліз сервісу, котрий міг би повністю відтворювати вже існуючий функціонал конкурентів та вдало доповнюючи його перечисленими інноваціями міг би бути широко популярним в населення. Вдалим рішенням є реалізація додатку саме на мобільну платформу, адже смартфон в наш час є невід'ємною частиною майже кожного жителя.

Застосунок має цілком відповідати поставленим функціональним та нефункціональним вимогам, вказаним в розділі.

РОЗДІЛ 2
МОДЕЛЮВАННЯ ТА АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Архітектура програмного забезпечення

Для того, щоб створити якісний програмний продукт, а саме мобільний додаток, потрібно серйозно підійти до вибору архітектури програмного забезпечення. В світі розробки мобільних додатків виділяють три основні архітектурні паттерни: MVC, MVP та MVVM. Розберемо більш детально кожен з них.

MVC(Model-View-Controller, українською – Модель-Вид-Контроллер) – один з найдавніших архітектурних паттернів. Базується на тому, щоб розподіляти всі частини ПЗ на три частини – модель, вид та контроллер.

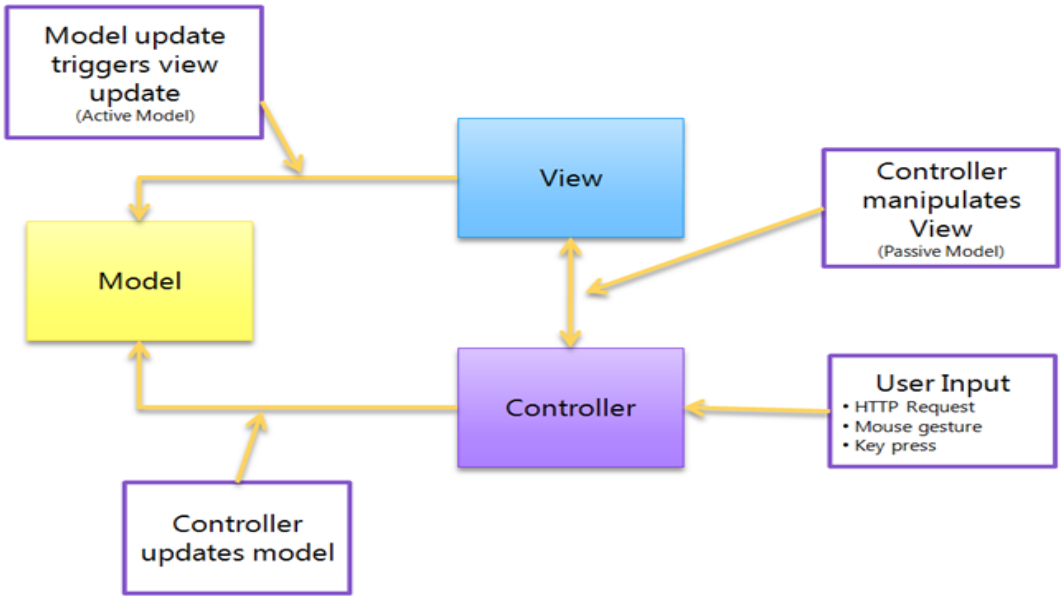


Рисунок 2.1 – Діаграма паттерну MVC[4]

Модель відповідає лише за данні і описує основний фунціонал їх обробки. Вона ніяким чином не причасна до процессу вводу/виводу.

Вид, або ж вигляд, відповідає за відтворення UI на екрані користувача.

Контроллер оброблює дії користувача і потім оновлює Модель або Вигляд. Якщо користувач взаємодіє з додатком (натискає кнопки на

клавіатурі, скролить контент), контролер отримує повідомлення про ці дії і вирішує, що з ними робити.

Головним недоліком паттерну є нагромадлення реалізації бізнес логіки та логіки по вводу/виводу в контролері. Як результат – з часом в цих класах кількість коду стає дуже високою, що сильно ускладнює підтримку такого коду, унеможливлює процес автоматизації тестування програмного забезпечення.

Після того, як розробники зрозуміли, що MVC як модель архітектури устаріла, а технології еволюціонували до того, що відпала необхідність в створенні власного, кастомного рішення дефолтних елементів, таких як кнопки, поле виводу тексту і тд на користь вбудованих рішень в фреймворки, розробники почали переходити на більш нову та абстрактну модель – MVP(Model-View-Presenter, українською – Модель-Вид-Презентер). Цей патерн описує, яким чином можна відділити логіку інтерфейсу від самого UI та окремо від даних.

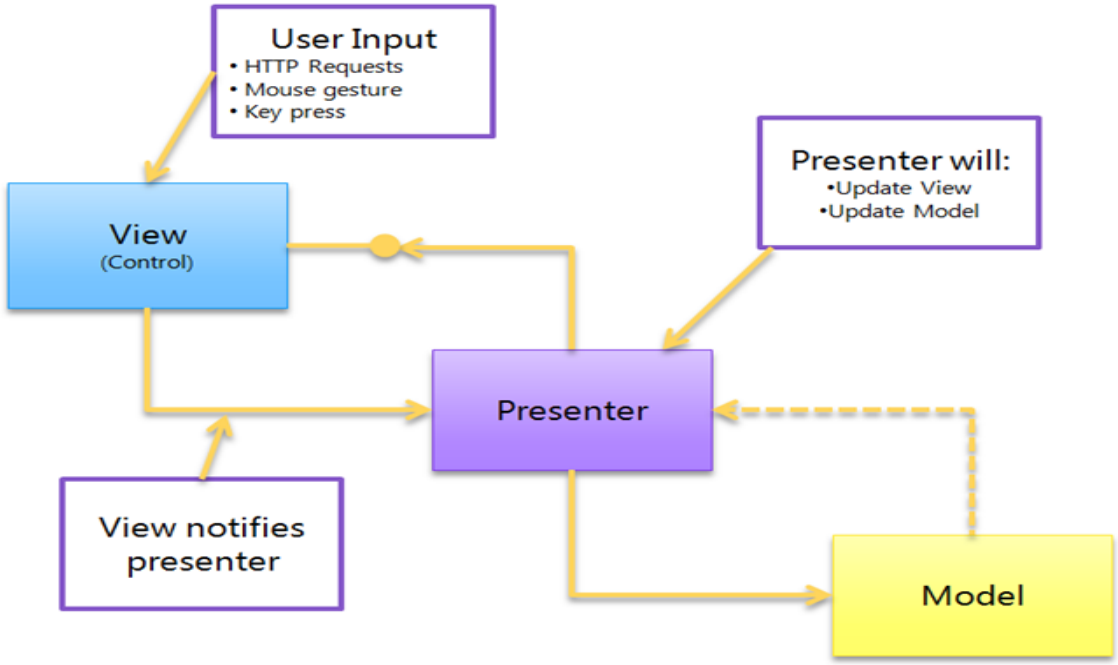


Рисунок 2.2 – Діаграма патерну MVP[4]

Як і в MVC, Модель в цьому паттерні відповідає за данні, а саме їх зберігання та отримання.

Вид(представлення) - зазвичай являє собою форму з віджетами. Користувач може взаємодіє з її елементами, але коли якась подія віджета зачіпає логіку інтерфейсу, Вид буде направляти його презентеру.

Презентер містить всю логіку, яка є в інтерфейсу і відповідає за синхронізацію моделі данних і її представлення. Коли вид повідомляє презентер, що користувач щось зробив (наприклад, змінив текс), презентер приймає рішення про оновлення моделі і синхронізує всі зміни між моделлю і представленням.

Однією з головних речей є те, що презенетер ніколи не взаємодіє з видом напряму. Для взаємодії використовують інтерфейси, що дозволяє тестування кожного з елементів поокремо.

Найбільш новітнім архітектурним паттерном є MVVM(Model-View-ViewModel, українською – Модель-Вид-МодельВиду). Даний патерн відрізняється від MVC та MVP логікою розподілу відповідальностей між модулями.

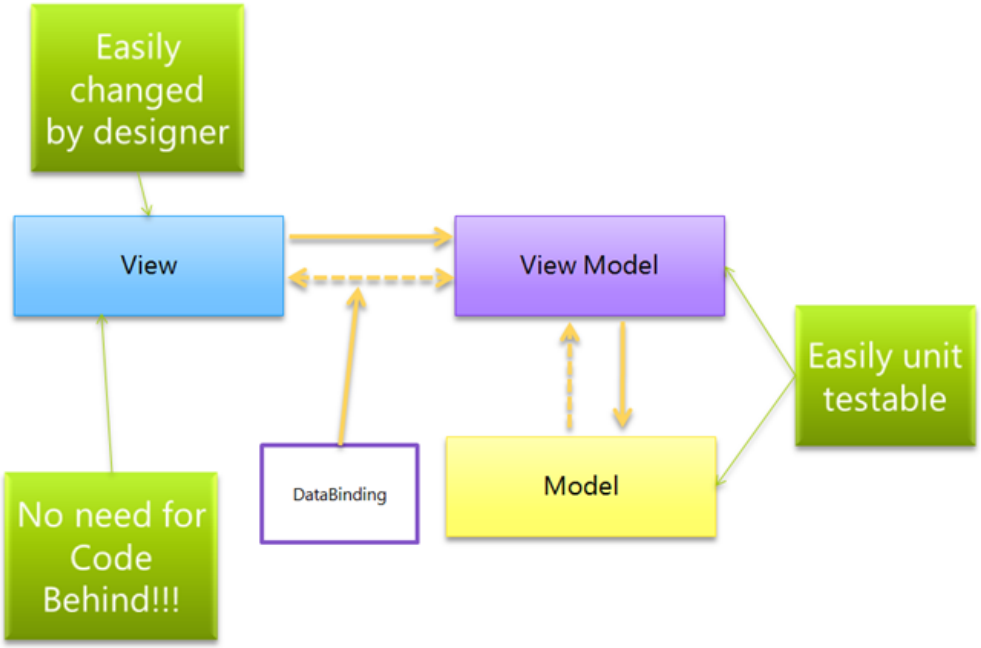


Рисунок 2.3 – Діаграма патерну MVVM[4]

Model - цей модуль не відрізняється від аналогічного в MVC. Він відповідає за створення моделей даних і може містити в собі бізнес-логіку.

View – модуль, котрий містить основні відмінності від інших двох паттернів. Модуль View в MVVM охоплює інтерфейс (класи UIView, файли xib/storyboard), логіку відображення (показ даних на екрані, анімацію UI елементів) і обробку призначених для користувача подій (натискання кнопок і т.д.) У MVC за це відповідають View і Controller. Це означає що інтерфейси у вас залишаться незмінними, в той час як на ViewController делегується менше відповідальності. Як результат, він стане набагато стислішим.

ViewModel – модуль, котрий буде містити всю бізнес догуку та логіку UI частини, котра раніше знаходилася у ViewController. Модуль запрошує дані у Моделі (це може бути запит до локальної базі даних або мережевий запит) і передає їх назад у Вид, вже в тому форматі, в якому вони будуть там використовуватися і відображатися. Але це двонаправлений механізм, дії або дані, що вводяться користувачем проходять через ViewModel і оновлюють Model. Оскільки ViewModel стежить за всім що відображається, то корисно використовувати механізм зв'язування між цими двома шарами. Саме цюямодель буде використана під час написання прощрамного забезпечення.

Програмне забезпечення буде написане на нативній мові програмування задля максимальної продуктивності і швидкості роботи застосунку. Нативний додаток може отримати повний доступ до всіх можливостей телефона без проблем, котрі зазвичай є у кросс-платформенних рішень. На даний момент є дві мови, котрі використовуються для розробки нативних рішень для платформи iOS – Objective-C та Swift. Objective-C вважається застарілим підходом, все більше застосунків, написаних на ній переходять на Swift через високу кількість коду, котрий тягне за собою икористання цієї мови. На противагу цьому, Swift пропонує більш лаконічний та зрозумілий підхід до написання ПЗ. Варто зазначити, що за підтримку та прогрес мови Swift

відповідає компанія Apple, тому що це їх продукт. На даний момент Swift пережив 5 основні ревізії, с кожною ревізією мова набуває все більше корисних фіч.

Так як архітектура MVVM базується на звязуванні данних, очевидно, що нам треба якимось чином реалізувати процес связування. Саме тому в цьому випадку доцільно використовувати RxSwift. Цей фреймворк дійсно відіграє велику роль у розробці додатків для iOS. RxSwift – імплементація ReactiveX API на платформу Swift для асинхронного програмування з потоками даних. Потрібно розуміти, що реактивне програмування – це така парадигма, котра спирається на потоки данних та змінення їх стану. Поток - це послідовність подій, котрі виуонуються з плином часу. Поток може повертати три види даних: якесь значення, помилку чи сигнал завершення події. Сигнал завершення розповсюджується, коли потік закінчує виконуватися.

Для зберігання даних в додатку доцільно використати фреймворк Realm та його реактивну реалізацію RxRealm. Realm - мобільна база даних для , яка доступна на Objective-C й Swift. Realm була створена, щоб стати краще і швидше, ніж SQLite і Core Data. Вона не тільки краще і швидше, але й більш проста та лаконічна у використанні. Realm є абсолютно безкоштовною. Realm створена для мобільних застосунків. Realm розроблена, щоб бути простою у використанні, так як вона не ORM, і вона використовує свій власний механізм персистентності для більшої продуктивності і швидкості виконання. Завдяки підтримці команди ReactiveX фреймворку Realm, користувачам доступна реактивна оболочка RxRealm, котру можна використовувати в нами обраній архітектурі. [2]

В якості сервера було обране готове рішення від компанії Google, а саме Firebase Database. По суті, Firebase є безумовно приголомшливим у виконанні, реалізації та експлуатації. Firebase служить базою даних, яка змінюється в реальному часі і зберігає дані в JSON. Будь-які зміни в базі даних відразу синхронізуються між усіма клієнтами, або девайсами, які використовують

одну і ту ж базу даних. Іншими словами, оновлення в Firebase відбуваються миттєво.

Разом зі сховищем, Firebase також надає призначену для користувача аутентифікацію, і тому всі дані передаються через захищене з'єднання SSL. Ми можемо вибрати будь-яку комбінацію email і пароля для аутентифікації, будь то Facebook, Twitter, GitHub, Google, або щось інше. [3]

2.2. Моделювання та аналіз програмного забезпечення

В даному підрозділі детально та поступово опишемо кожен з можливих варіантів взаємодії з додатком.

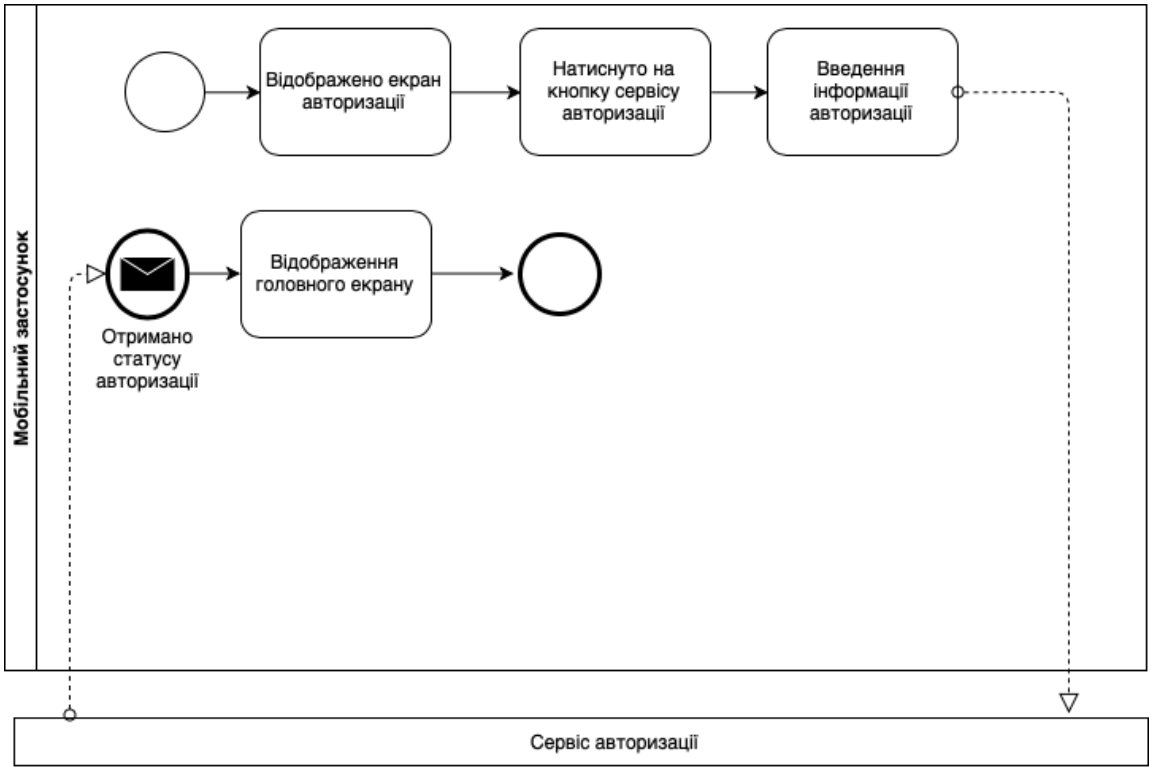


Рисунок 2.4 – Діаграма процесу авторизації

Опис авторизації в застосунку:

- Користувач переходить на екран з авторизацією.
- Користувач натискає на один з представлених методів авторизації. Користувач вводить данні для авторизації на тому сервісі, котрий він обрав.
- Якщо користувач правильно ввів данні для авторизації, додаток отримує повідомлення про успішну авторизацію користувача через додаток та отримує інформацію про користувача
- Додаток переводить користувача на головний екран

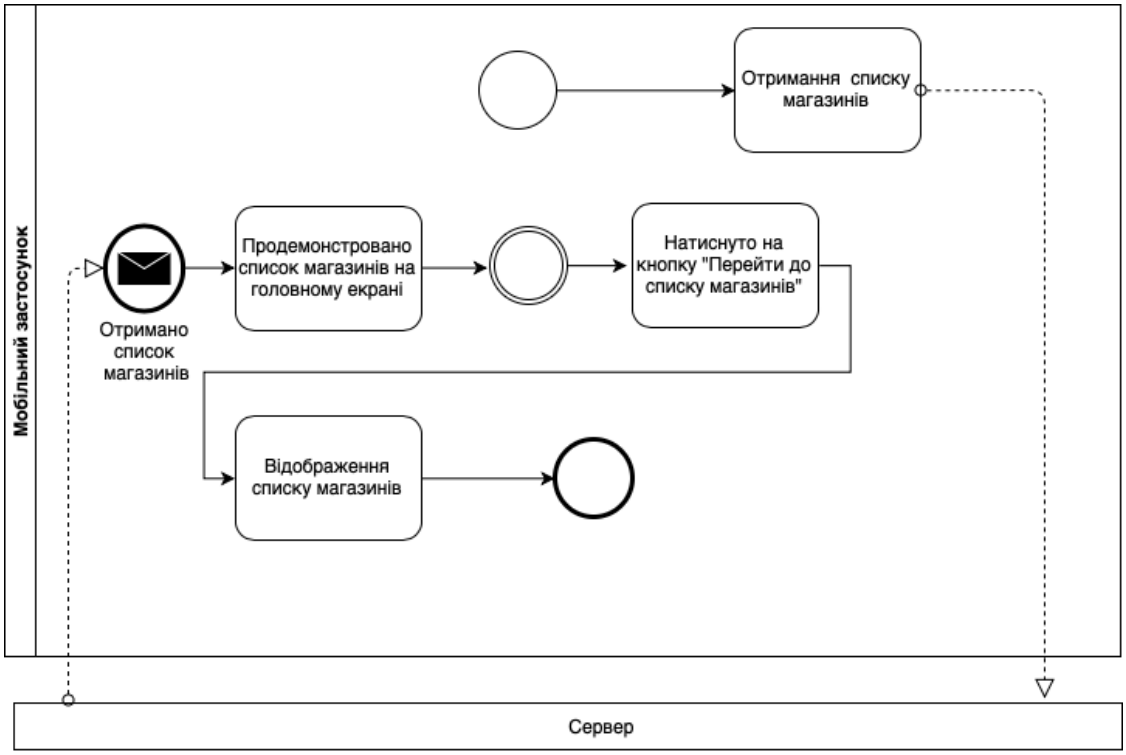


Рисунок 2.5 – Діаграма процесу отримання магазинів

Опис отримання магазинів:

- Користувач авторизований та знаходиться на головній сторінці на вкладці “Дім”.
- Мобільний застосунок робить виклик на бекенд для отримання списку магазинів. Після успішного отримання магазинів додаток відображає горизонтальний список з магазинами у секції з магазинами.
- В секції з магазинами є кнопка перейти до списку магазинів. Користувач натискає цю кнопку, додаток перенаправляє користувача на екран з вертикальним списком магазинів та полем пошуку магазину по його назві.
- Якщо користувач вводить назву магазину в поле пошуку, застосунок фільтрує список магазинів по введеному слову та видає лише підходящі варіанти.

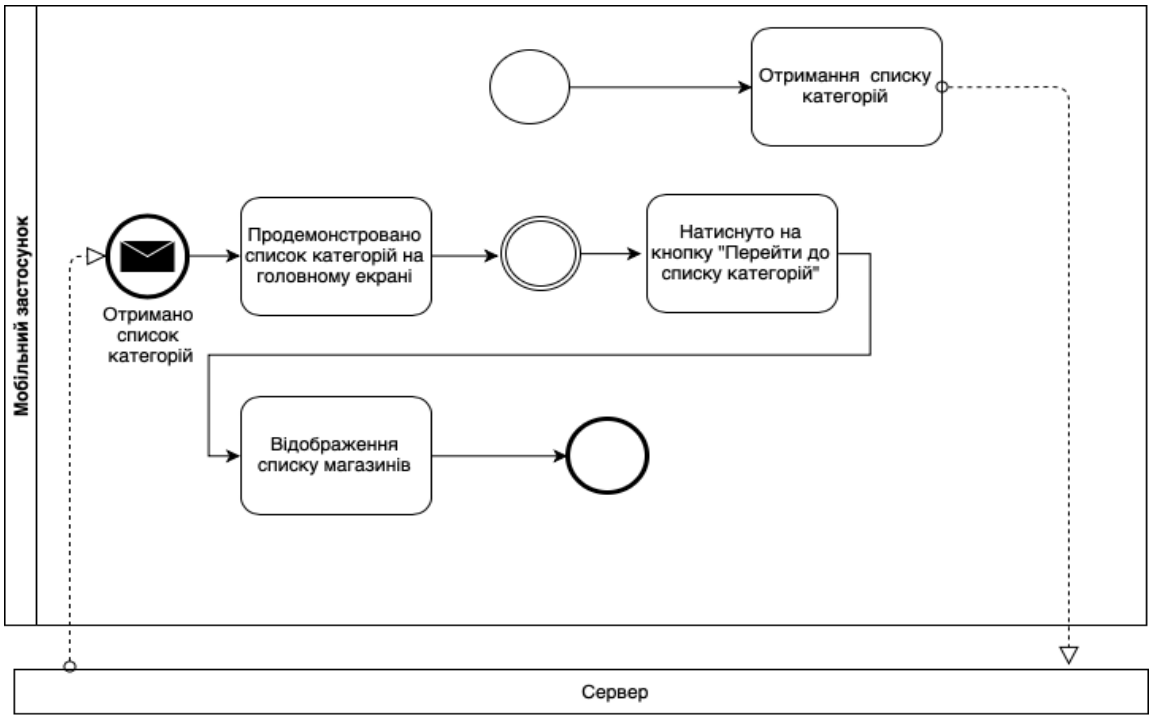


Рисунок 2.6 – Діаграма процесу отримання категорій товарів

Опис отримання категорій товарів:

- Користувач авторизований застосунок успішно отримав список магазинів, категорій та товарів з серверу.
- Мобільний застосунок робить виклик на сервер для отримання списку всіх категорій. Після отримання списку категорій товарів застосунок відображає отримані дані у вигляді горизонтального списку категорій у секції з категоріями на головному екрані.
- В секції з категоріями є кнопка перейти до списку категорій. Користувач натискає цю кнопку, додаток перенаправляє користувача на екран з вертикальним списком категорій та полем пошуку категорії по її назві.
- Якщо користувач вводить назву категорії в поле пошуку, застосунок фільтрує список категорій по введеному слову та видає лише підходящі варіанти.

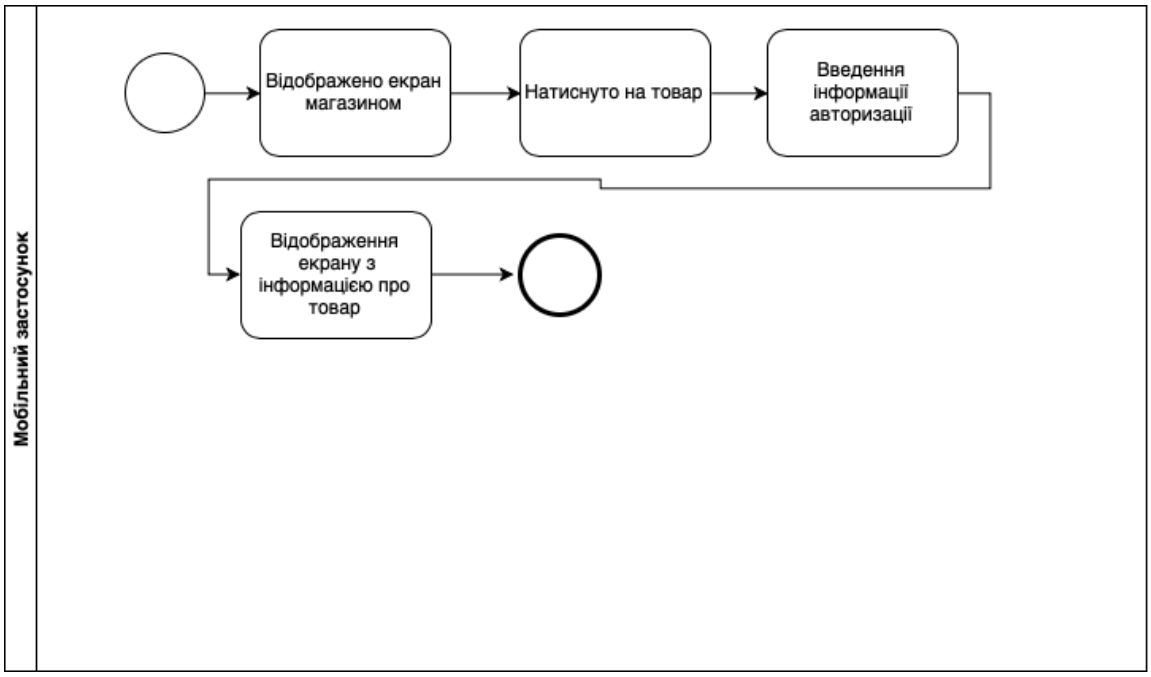


Рисунок 2.7 – Діаграма процесу отримання інформації про товар

Опис отримання інформації про товар:

- Користувач авторизований застосунок успішно отримав список магазинів, категорій та товарів з серверу.
- Користувач натискає на товар на вкладці “Дім” або в списку категорій товарів
- Застосунок перенаправляє користувача на екран з детальною інформацією про товар

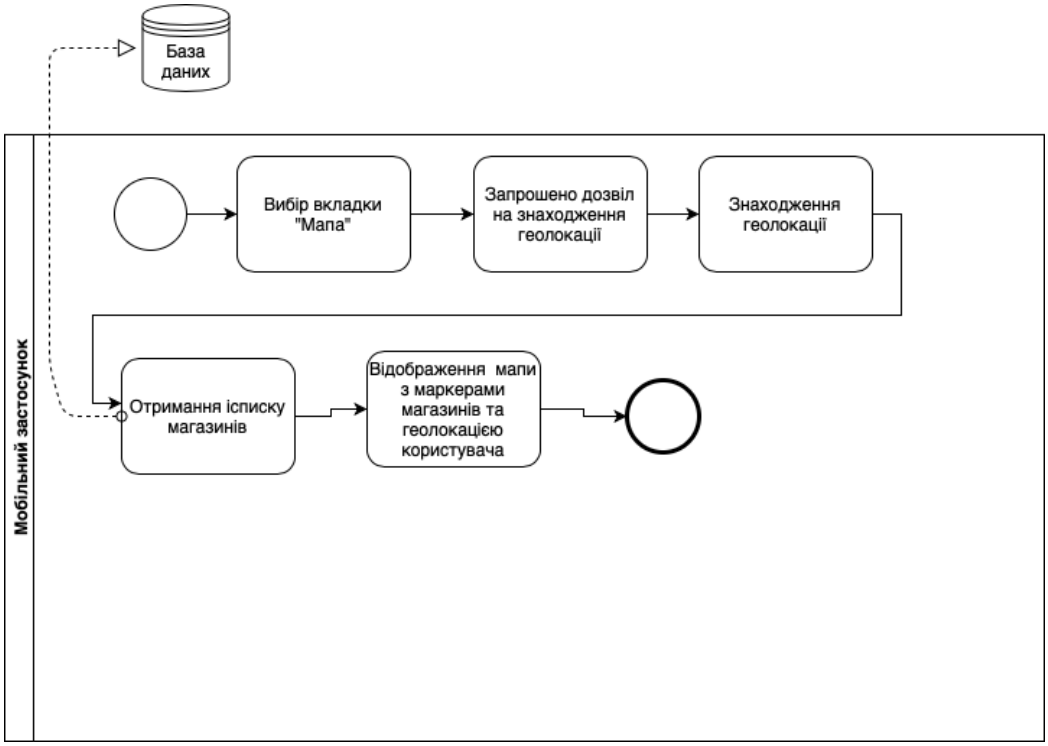


Рисунок 2.8– Діаграма процесу відображення мапи

Опис відображення магазинів на мапі:

- Користувач авторизований.
- Користувач натискає вкладку “Мапа”
- Застосунок пропонує користувачеві надати дозвіл на надання геолокації. Користувач надає доступ
- Застосунок відображає мапу, на якій знаходяться маркери з магазинами та маркер, на якому відображена позиція користувача
- Якщо користууувач натискає на маркер з магазином, застосунок показує скорочену інормацію про магазин в вікні знизу вкладки “Мапа”. Якщо користувач натискає на цю інформацію, застосунок перенаправляє користувача на екран з інформацією про магазин та списком товарів і категорій товарів цього магазину.

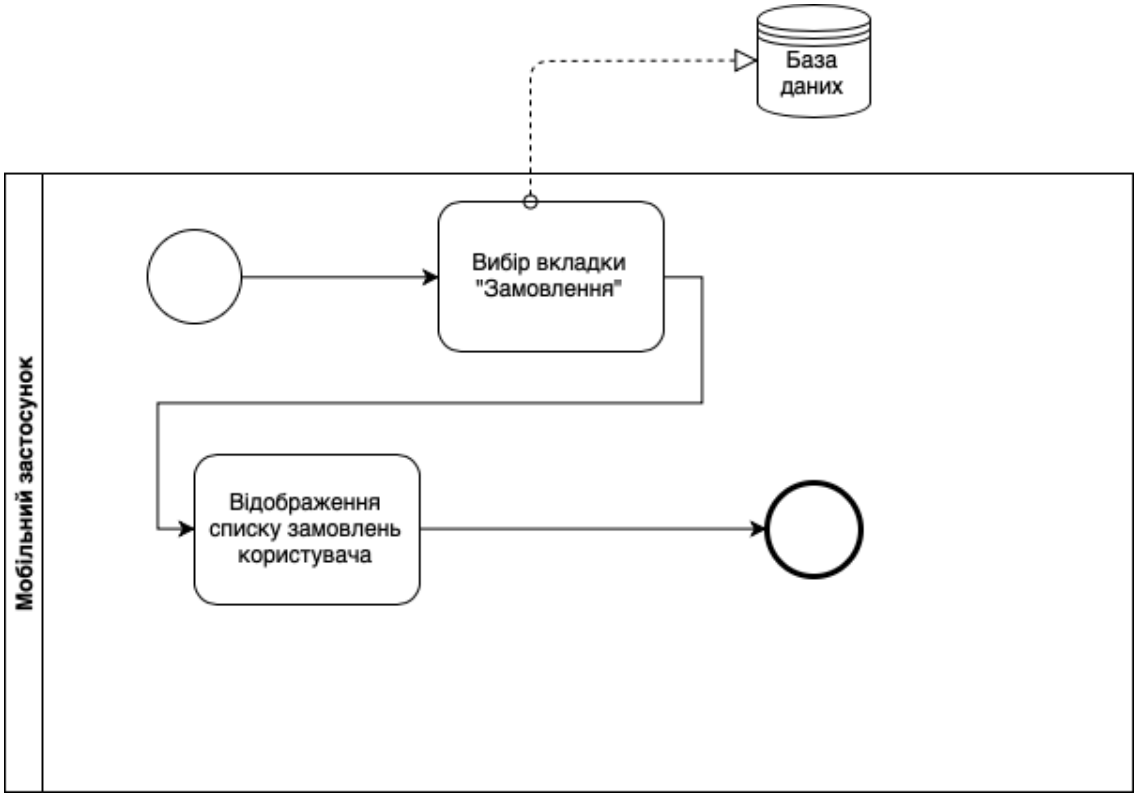


Рисунок 2.9 – Діаграма процесу відображення замовлень

Опис відображення замовлень користувача:

- Користувач авторизований.
- Користувач натискає вкладку “Замовлення”
- Застосунок відображає список замовлень користувача, в кожному елементі якого є інформація про суму замовлення та відображені товари, які придюав користувач в замовленні

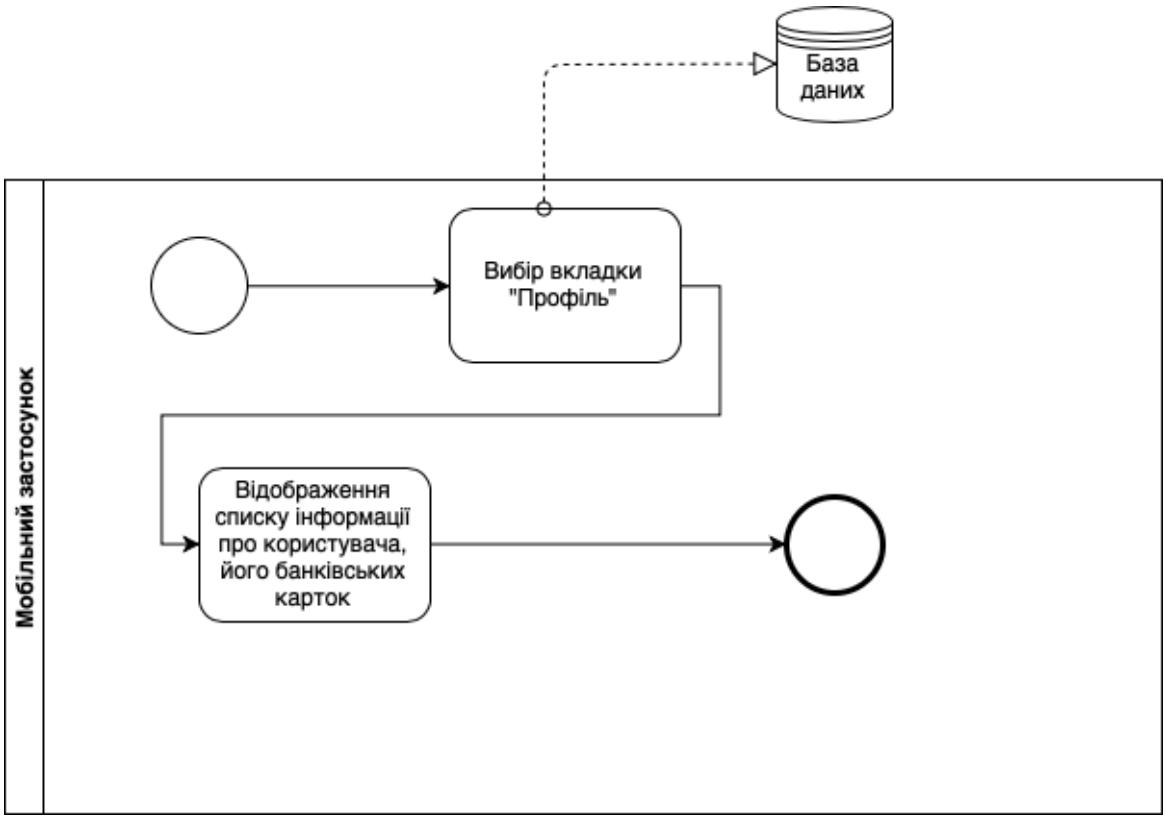


Рисунок 2.10– Діаграма процесу відображення профілю

Опис відображення інформації користувача:

- Користувач авторизований.
- Користувач натискає вкладку “Профіль”
- Застосунок відображає інформацію про ім’я, електронну пошту та номер телефону користувача. Також застосунок відображає список доданих банківських карт користувача

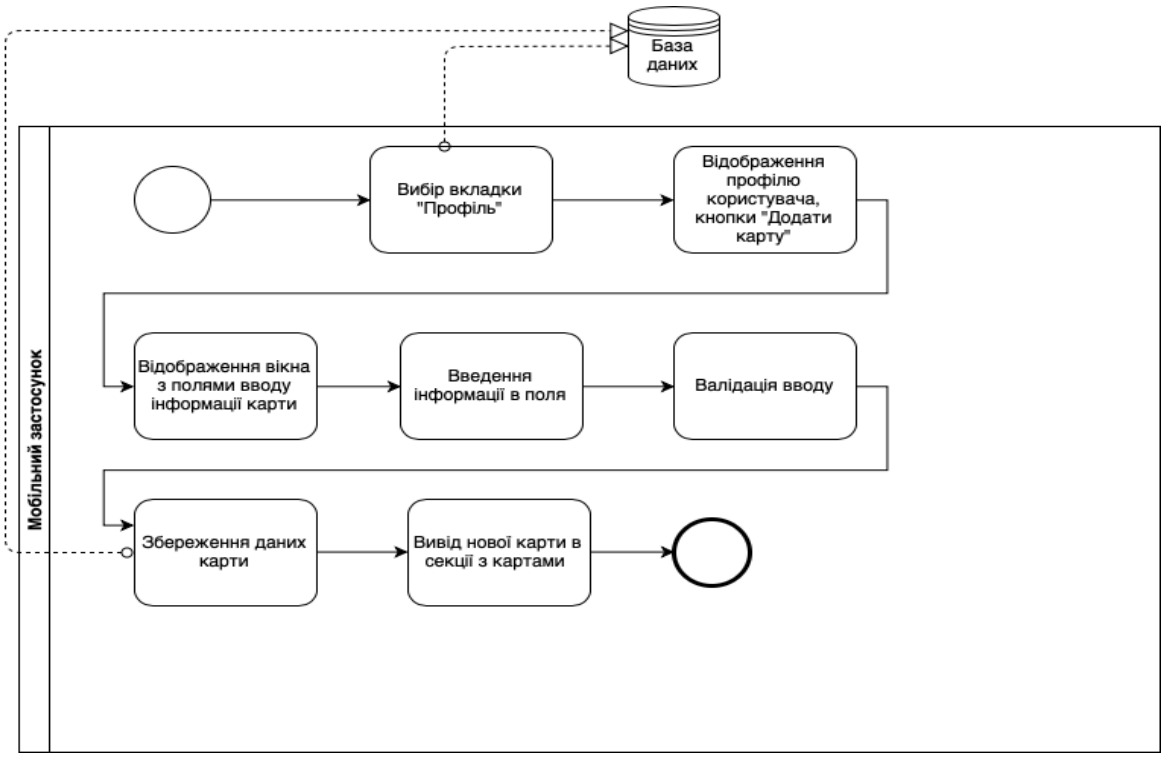


Рисунок 2.11 – Діаграма процесу додавання банківської карти

Опис додавання карти користувача:

- Користувач авторизований.
- Користувач натискає вкладку “Профіль”
- Застосунок показує профіль користувача та кнопку додати карту
- Користувач натискає цю кнопку
- Застосунок презентує модальне вікно з полями “Назва карти”,”Номер карти”,”Місяць карти”,”Рік карти”,”CVC номер”,”Володар карти” та кнопками “Скасувати” і “Додати карту”
- Користувач вводить всю інформацію та натискає кнопку додати.
- Застосунок валідує всі поля. Якщо введені корректні данні, застосунок закриває вікно та додає введену карту в список карт користувача.
- Якщо введені дані не є коректними додаток не дає змогу додати цю карту.

- Якщо користувач натискає кнопку “Скасувати” додаток закриває вікно і не додає карту

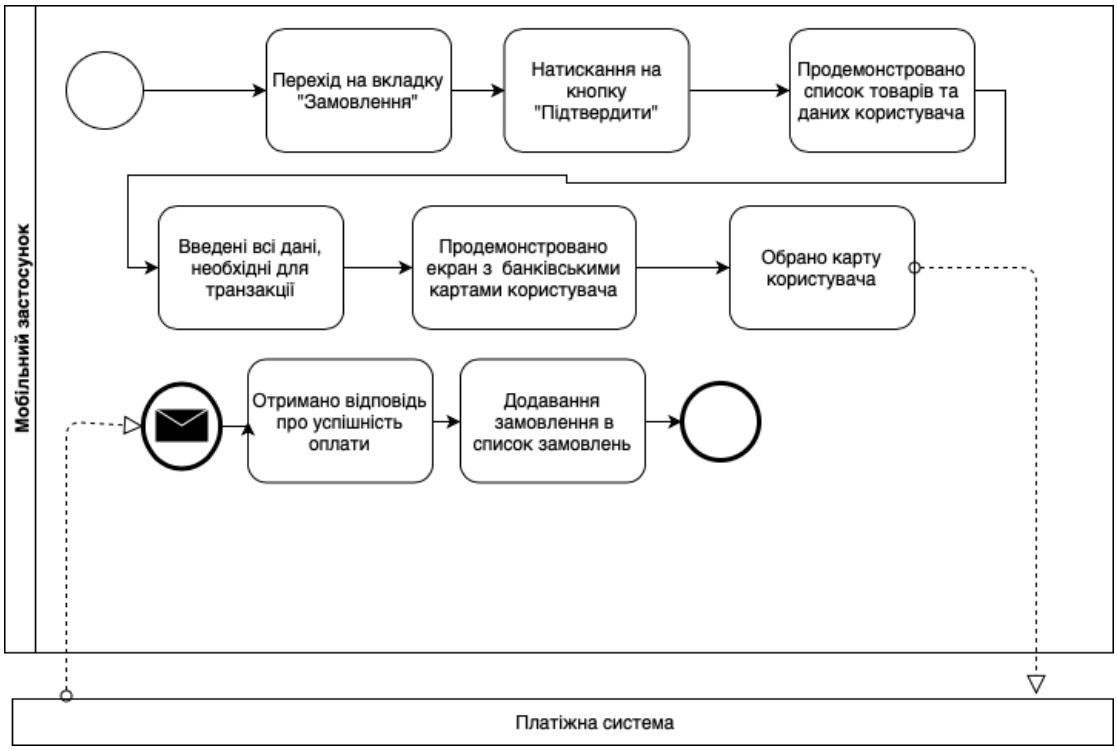


Рисунок 2.12 – Діаграма процесу оформлення замовлення

Опис оформлення замовлення товару/товарів з магазину:

- Користувач авторизований.
- Користувач додав товар або групу товарів з магазину
- Користувач переходить на вкладку “Замовлення”
- Застосунок відображає список замовлень в магазинах, котрих користувач додав товар
- Користувач натискає кнопку “Підтвердити”
- Застосунок перенаправляє користувача на екран з списком товарів з магазину, котрі додав користувач в корзину та полями з ім’ям користувача,його номером телефону та інтерактивної мапи.

Користувач заповнює поля, котрі є незаповненими, на мапі обирає адрес доставки

- Після того, як вся інформація введена, застосунок дає можливість користувачеві натиснути кнопку “Оплатити”
- Користувач натискає на кнопку
- Застосунок перенаправляє користувача на екран, де він може обрати вже додану або додати банківську карту
- Користувач натискає на карту, за допомогою якої він хоче здійснити оплату замовлення
- Застосунок передає всю інформацію в платіжну систему та викликає метод оплати
- Платіжна система показує екран з конфірмацією оплати
- Якщо користувач конфірмував оплату, застосунок показує діалгове вікно з інформацією, що оплата пройшла успішно. Користувач натискає кнопку “Закрити”, додаток перенаправляє користувача на головний екран.
- Застосунок додає оплачене замовлення користувача в в вкладку “Замовлення”

Структура моделей даних, яка використовується в застосунку, описана в таблиці 2.1.

Таблиця 2.1 – Структури моделей даних

Store	{id: String, merchantId: String, name: String, photoUrl: String, address: String, description: String, longitude: Double, latitude: Double, startWorkingSession: Int, endWorkingSession: Int, blocked: Bool, categories: Array<StoreCategory> }	Структура магазину. Містить в собі id магазину, назву, посилання на картинку магазину, адрес, широту та довготу для відображення на мапі, початок та закіечення робочого дня а також список категорій, під які підпадає магазин

Category	{
	storeId: String,
	categoryId: String,
	name: String,
	photoUrl: String,
	blocked: Bool
	}
		Категорія товару. Мвстить в собі ідентифікатор магазину, в якому вона знаходиться, власний ідентифікатор, назву, посилання на фотографію категорії а також статус, котрий відповідає за статус заблокованості

Продовження таблиці 2.1

Stuff	{id: String, storeCategoryId: String, stuffCategoryId: String, parentStoreId: String, name: String, price: String, photoUrl: String, description: String, blocked: Bool, }	Структура, що описує товар. Містить в собі id магазину, ідентифікатор категорії товару, назву, ціну, опис товару , посилання на картинку магазину.

Card	{
	number: String,
	expMm: Int,
	expYy: Int,
	cvv: String,
	label: String,
	owner: String
	}
		Банківська карта. Містить в собі номер карти, місяць та рік, по яку вона гідна, CVV код, ім’я та призвіще господаря та назву карти.

2.3. Аналіз безпеки даних

Виділимо основні вектори атак, котрі можуть проводитись на даний мобільний застосунок – це перехоплення данних між сервером та застосунком й можливість дослідження і взлому функціоналу, пов’язаним з платіжною системою.

Так як в якості сервера використовується рішення від компанії Google, а саме Firebase Database, то за безпеку передачі даних відповідає їх SDK. В Firebase всі дані передаються через SSL(Secure Sockets Layer) - це стандартна

технологія захисту для встановлення зашифрованого зв'язку між сервером і клієнтом - зазвичай це сервер та браузер/мобільний додаток.

SSL дозволяє надійно передавати конфіденційну інформацію, наприклад номери кредитних карток, номери соціального страхування та облікові дані для входу. Зазвичай дані, що передаються між користувачем та сервером, надсилаються у вигляд Plain - тексту, що робить вас вразливими до того, що вашу дату можна відобразити без процесу декодингу.

SSL є протоколом безпеки. SSL визначає змінні шифрування як для зв'язку, так і для переданих даних.

Другим вектором атак, котрі можуть проводитись на додаток, є можливий взлом функціоналу, пов’язаного з оплатою товару. В даному застосунку була використана платіжна система Fondy. По-перше, система Fondy підтримує можливість проведення 3DSecure платежів, котрі за своєю суттю прирівнюються до введення пін-коду. 3DSecure – протокол, котрий реалізує аутентифікацію користувача в дві фази. Працює він таким чином, що використовує банк-емітент з ціллю надсилання додаткового PIN-коду на номер телефону користувача і надає вікно для вводу цього поля користувачу. В такому випадку, коли шахрай заволодів картою, він все одно не зможе використати її дані для оплати чого-небудь, адже йому потрібно знати код, котрий прийде на телефон персоні, в кого він викрав карту.

По-друге, власна система Antifraud попереджує можливість махінацій, шляхом фільтрування підозрілих дій. Вона містить більше двухсот різноманітних інструкцій, за якими вона розпізнає підозрілі операції та блокує їх виконання. Як приклад, можна навести деякі з них:

- Історія попередніх покупок користувача
- Використання системи 3DSecure
- Перевірка країни, де зареєстрований власник карти та IP адресу людини, котра ініціювала замовлення з цієї карти

- Підозріла активність, котра може бути виявленна в великій кількості платежів за відносно короткий проміжок часу

По-третє, система Fondy також використовує SSL протокол, описаний раніше.

ВИСНОВКИ ДО РОЗДІЛУ 2

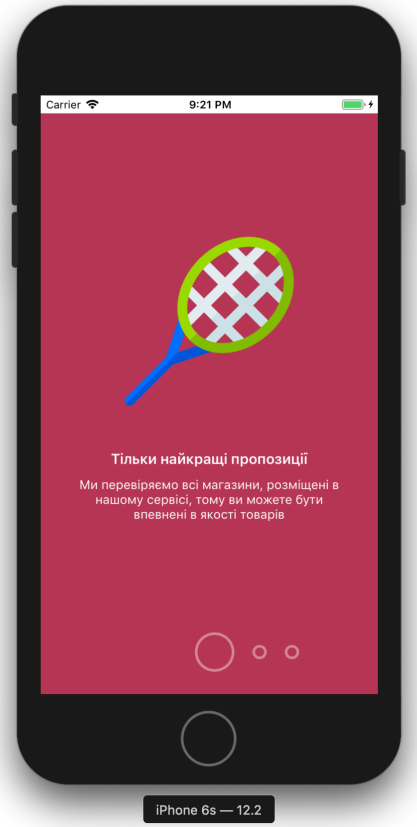
В цьому розділі було проведено аналіз технологій та патернів, котрі будуть використовуватися під час розробки. В якості архітектурного патерну буде використовуватися MVVM, база даних буде використана Realm, інтернет запити будуть реалізовані вбудованими в систему засобами. Додаток буде написаний на мові Swift.

Також були опрацьовані бізнес-процеси застосунку, графічно відображені за допомогою діаграм BPMN.

Були пропрацьовані основні можливі вектори атак на застосунок та вивчено методології їх зупинення та попередження.

РОЗДІЛ 3
ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1. Інструкція користувача



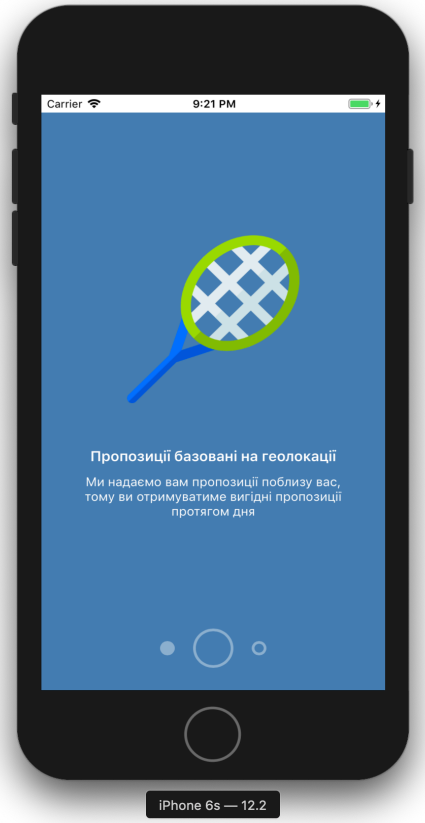


Рисунок 3.1 та Рисунок 3.2

-Перша та друга сторінки ознайомлення користувача з функціоналом застосунку

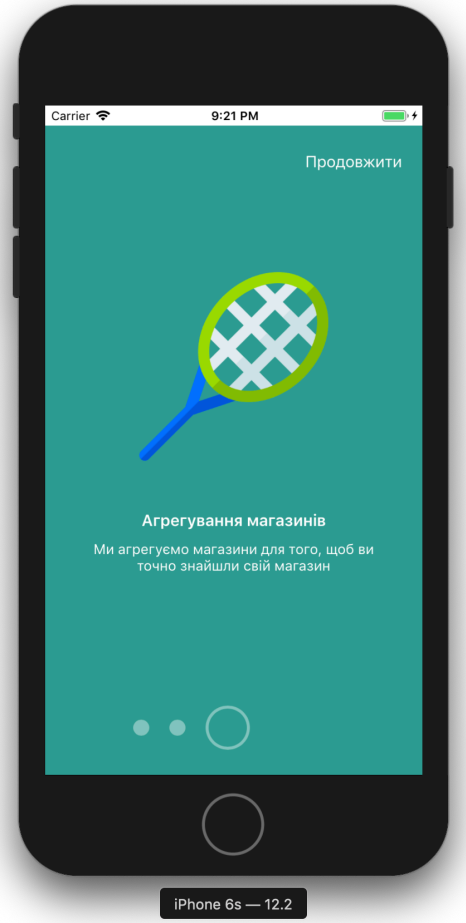


Рисунок 3.3 – третя сторінка ознайомлення користувача з функціоналом застосунку

Коли користувач тільки встановлює застосунок, його ознайомлюють з можливостями використання данного застосунку. Після того, як користувач натисне на кнопку “Продовжити”, застосунок переведе користувача на екран аутентифікації, де він може обрати один з двох методів аутентифікації – за допомогою аккаунту Google чи аккаунту Facebook. Цей екран зображений на рисунку 3.4

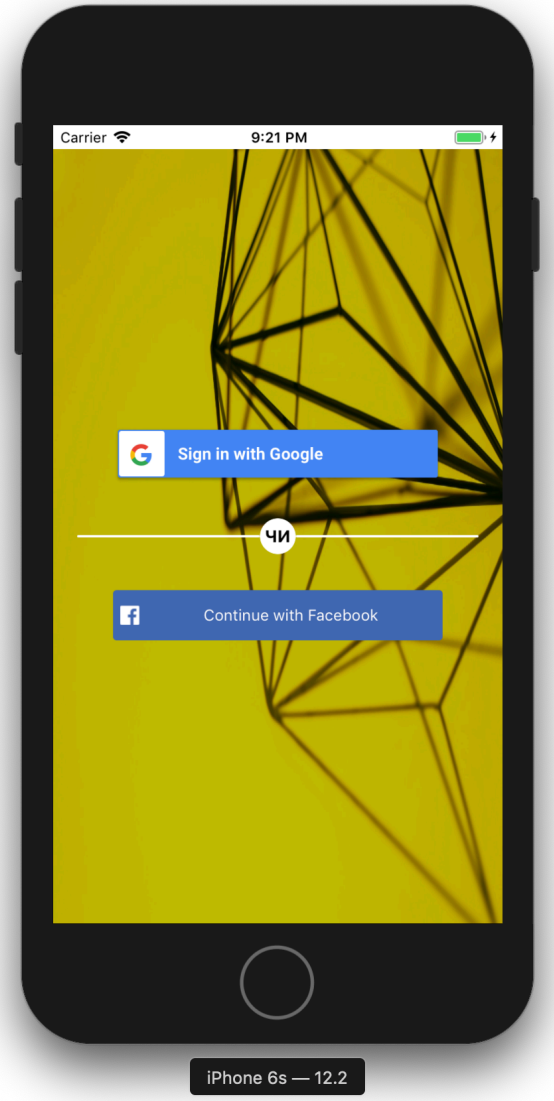


Рисунок 3.4 – екран вибору методу аутентифікації

В залежності від того, який метод аутентифікації обере користувач, він побачить один з екранів. В разі вибору аутентифікації через аккаунт Google, він побачить екран, зображений на рисунку 3.5, в випадку з аккаунтом аутентифікації через систему Facebook- екран на рисунку 3.6

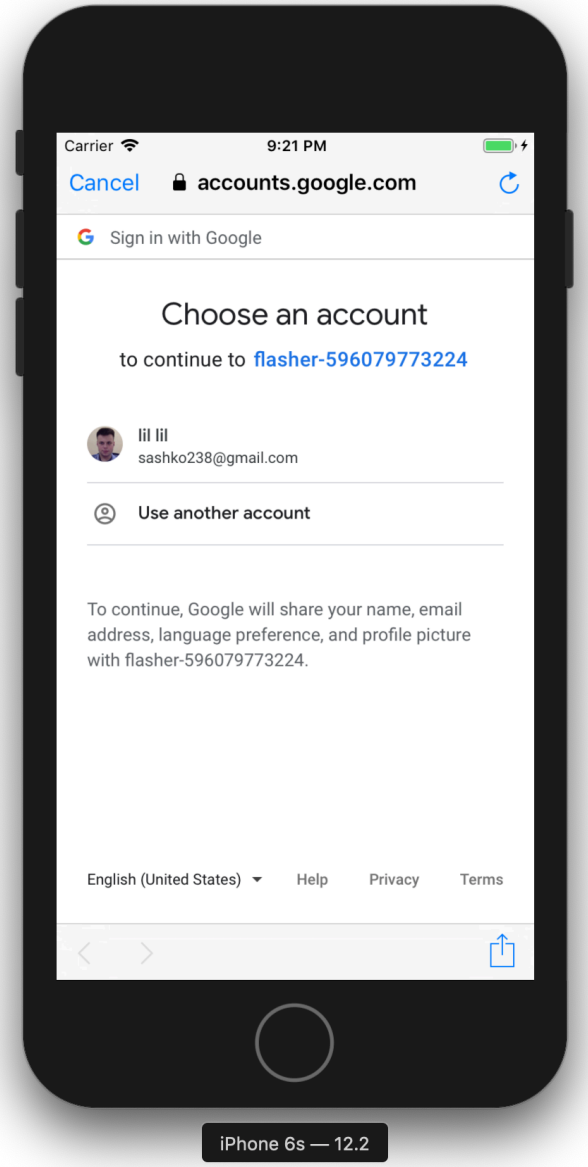


Рисунок 3.5 – екран аутентифікації через аккаунт Google

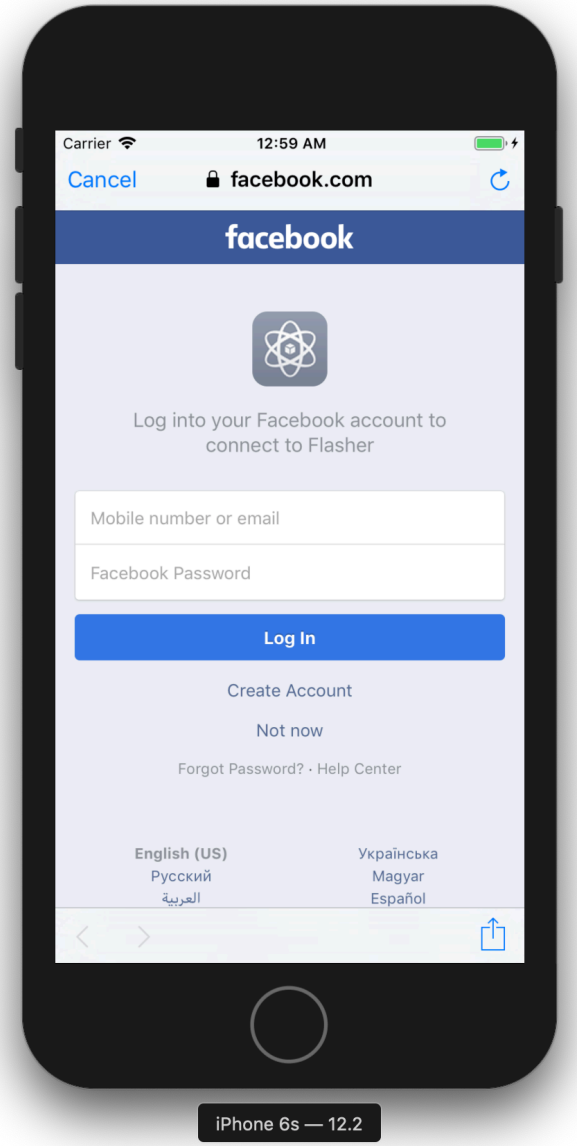


Рисунок 3.6 – екран аутентифікації через Facebook

Після того, як користувач успішно аутентифікувався, застосунок перенаправляє користувача на головний екран, котрий складається з таких вкладок: “Головна”,”Мапа”,”Кошик”,”Історія” та ”Профіль” з за замовчуванням відкритою вкладкою “Головна”(Рисунок 3.7)

Рисунок 3.7 – Головний екран, відкритий на вкладці “Головна”

Вкладка “Головна” складається з:

Слайдеру з спеціальними пропозиціями (зображений з самого верху рисунку 1.1.7)

- Вкладки з магазинами
- Вкладки з категоріями
- Вкладки з товарами в топі продаж

Якщо користувач натисне на товар з секції спеціальної секції, застосунок перенаправить користувача на сторінку товару, де користувач може побачити детальну інформацію про товар (Рисунок 3.9), його ціну та може натиснути кнопку “Придбати”, щоб додати товар в кошик.

Якщо користувач натисне на магазин в списку магазинів, застосунок перенаправить користувача на екран з детальною інформацією про магазин(Рисунок 3.10, Рисунок 3.11).

Якщо користувач обирає якусь з категорій товарів, застосунок відкриває екран з магазином, дн присутня ця категорія на саме цій категорії (Рисунок 3.11).

Якщо користувач натисне на товар в секції топ продаж (Рисунок 3.8), застосунок перенаправляє користувача на сторінку товару.

Якщо користувач натисне на іконку > в секції категорій, застосунок перенаправить користувача на екран з списком категорій товарів(Рисунок 3.12), де можна знайти категорію за назвою (Рисунок 3.13).

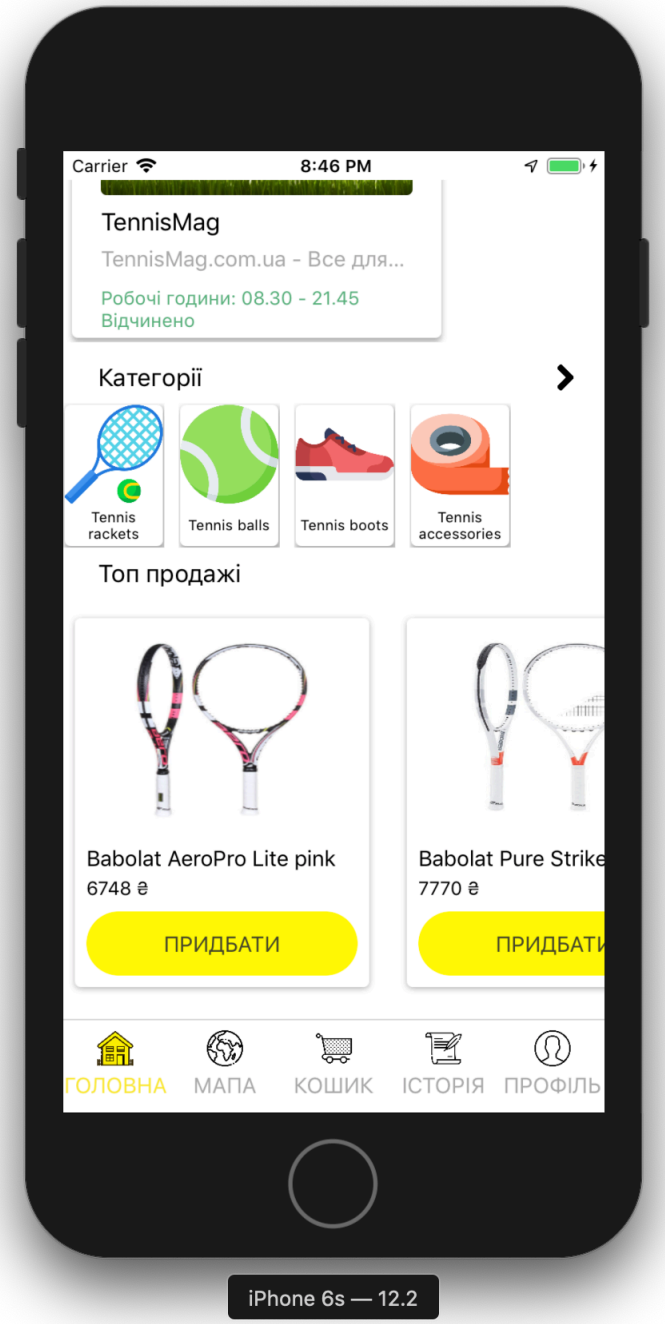


Рисунок 3.8 – Головний екран, відкритий на вкладці головна, проскролений до вкладки з топ продажі

Рисунок 3.9 – Екран з інформацією про товар

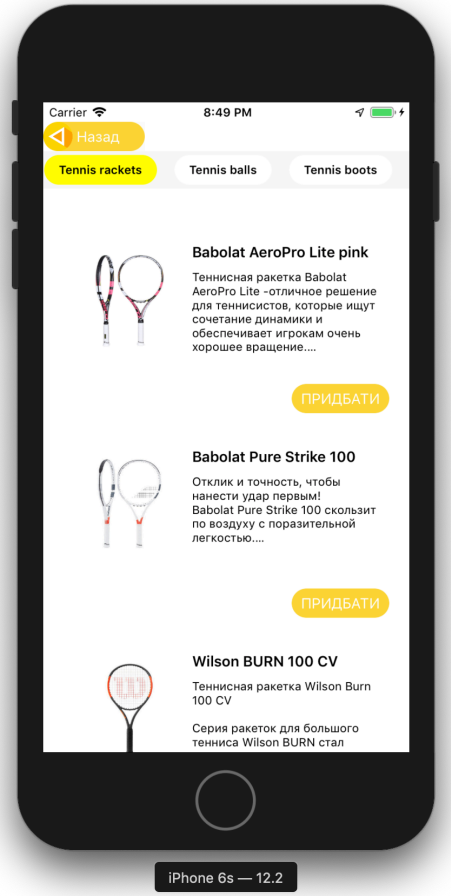
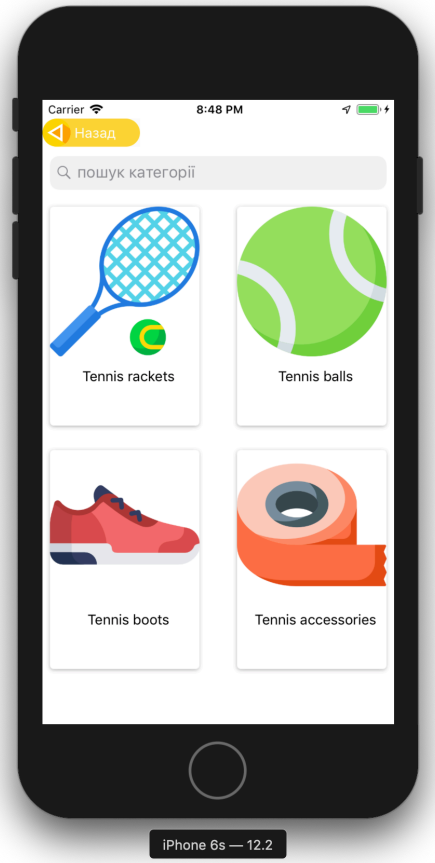


Рисунок 3.10 та Рисунок 3.11 – Екран з інформацією про магазин, його категоріями та товарами, представленими в ньому



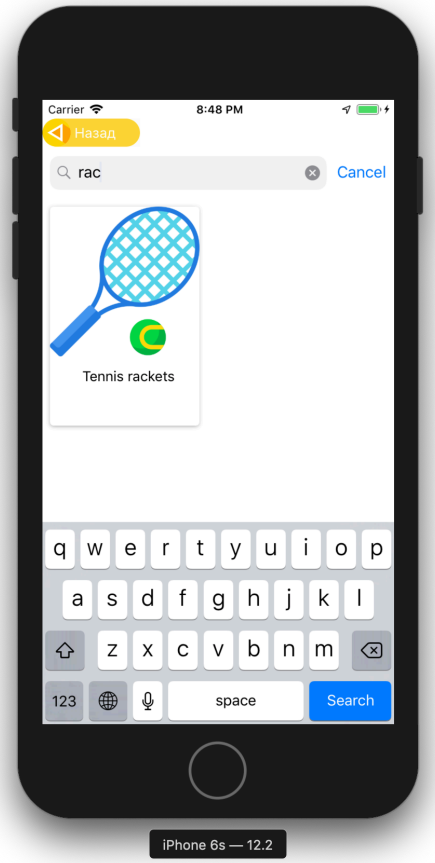


Рисунок 3.12 та Рисунок 3.13 – Екран з категоріями товарів, де можна здійснити пошук за назвою категорії

Коли користувач натискає на вкладку “Мапа”, застосунок відкриє вкладку з мапою. Буде зображено місцеположення користувача та маркери з кожним з магазинів. Якщо користувач натискає на маркер магазину, застосунок відображає коротку інформацію про магазин. Якщо користувач натисне на блок з інформацією про магазин, застосунок переведе користувача на екран з інформацією про магазин. Цей екран зображений на Рисунку 3.14

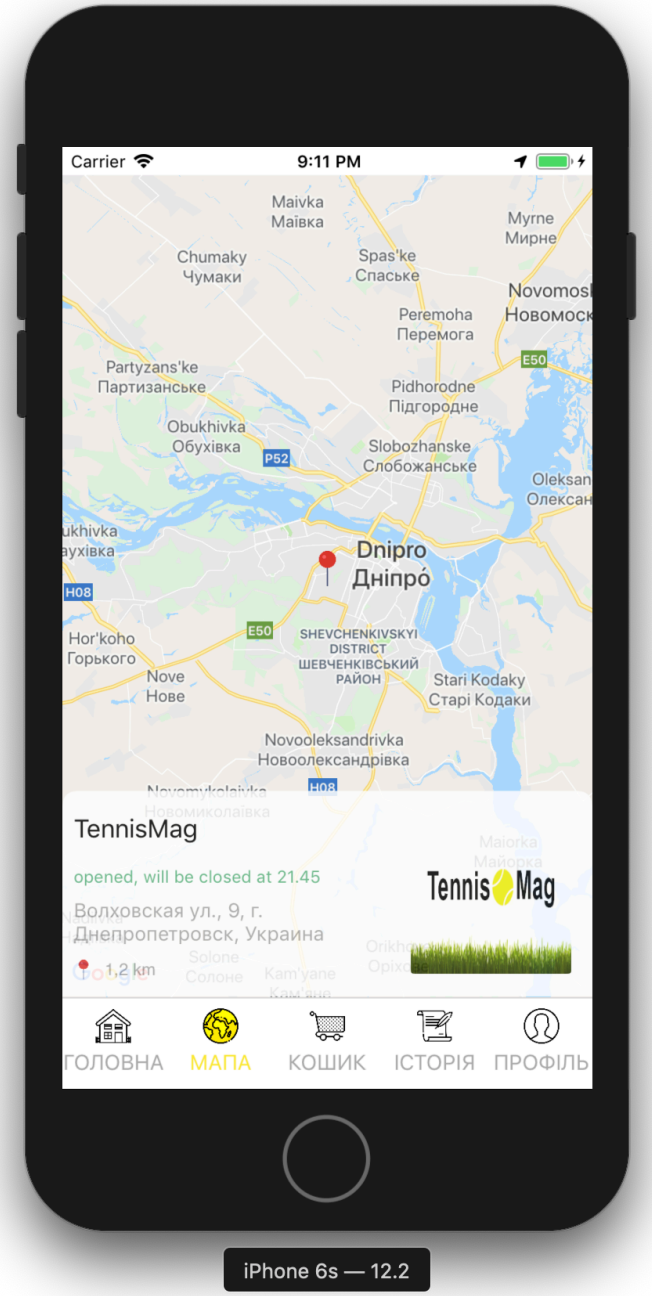


Рисунок 3.14 – Екран вкладкою “Мапа” та короткою інформацією про обраний магазин

Коли користувач натисне на вкладку “Кошик” застосунок відкриє екран з товаром або групою товарів відповідно кожного з магазинів, які користувач вже додав до кошику (Рисунок 3.15). В самому верху буде відображена загальна сума замовлень та кількість товарів в кошику для всіх магазинів, нижче буде відображена кількість товарів та сума замовлення відповідно до магазину. В кожному з товарів присутня можливість збільшити або зменшити кількість кожного з товарів. Якщо користувач натисне на кнопку “Оплатити”, застосунок відкриє екран з оформленням замовлення.

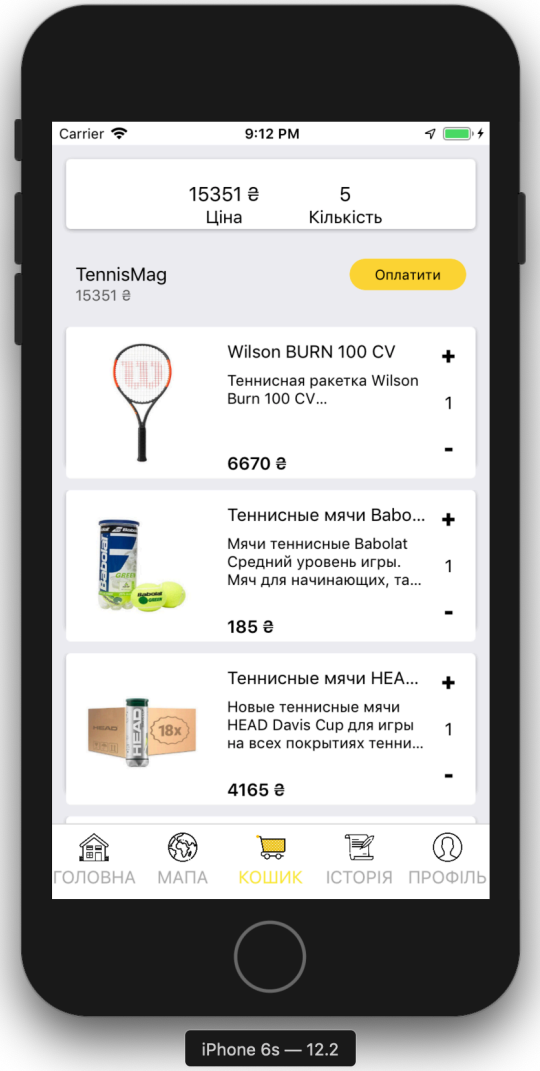


Рисунок 3.15 – Екран з кошиком товарів користувача

Коли користувач натисне на вкладку “Історія” застосунок відкриє екран з оплаченими замовленнями користувача (Рисунок 3.16). Кожне з замовлень містить в собі інформацію про назву магазину, оплаченій ціні та кількості придбаних товарів. Знизу відображається інформація про кожен з придбаних товарів – його назва, ціна та кількість придбаних одиниць товару.

Рисунок 3.16 – Екран з оплаченими замовленнями користувача

Коли користувач натисне на вкладку “Профіль” застосунок відкриє екран з профілем користувача (Рисунок 3.17). В цьому місці користувач може побачити інформацію про себе. Ця інформація буде використовуватися під час оформлення замовлення. Також на цьому екрані присутня інормація про вже додані банківські карти користувача, за допомогою яких він може здійснювати оплату замовлень. Якщо натиснути на кнопку “+”, користувач зможе додати ще одну банківську карту

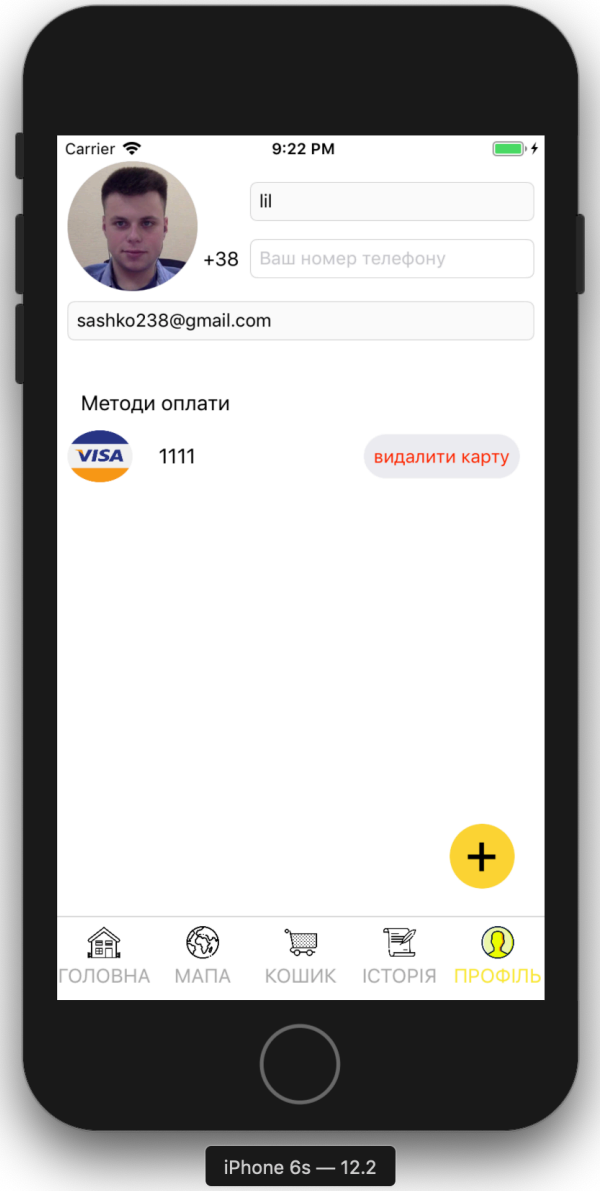


Рисунок 3.17 – Вкладка “Профіль”

Коли користувач натисне кнопку “Оплатити” на вкладці “Профіль” під обраним замовленням, застосунок відкриє вікно з верифікацією замовлення (Рисунок 3.18). В цьому місці користувач має заповнити незаповнені поля з деталями замовлення та обрати місце доставки товару. Також в цьому місці відображаються всі товари з замовлення.

Коли користувач натисне на кнопку “Оплатити”, застосунок перенаправить користувача на екран вибору платіжної карти, за допомогою якої користувач хоче здійснити оплату (Рисунок 3.19). Також на цьому екрані можна додати ще одну банківську карту (Рисунок 3.20).

Коли користувач натискає на банківську карту, застосунок передає її дані в систему оплати. Система оплати показує вікно верифікації оплати.

Після того, як платіжна система верифікувала оплату, застосунок додає замовлення в вкладку “Історія”

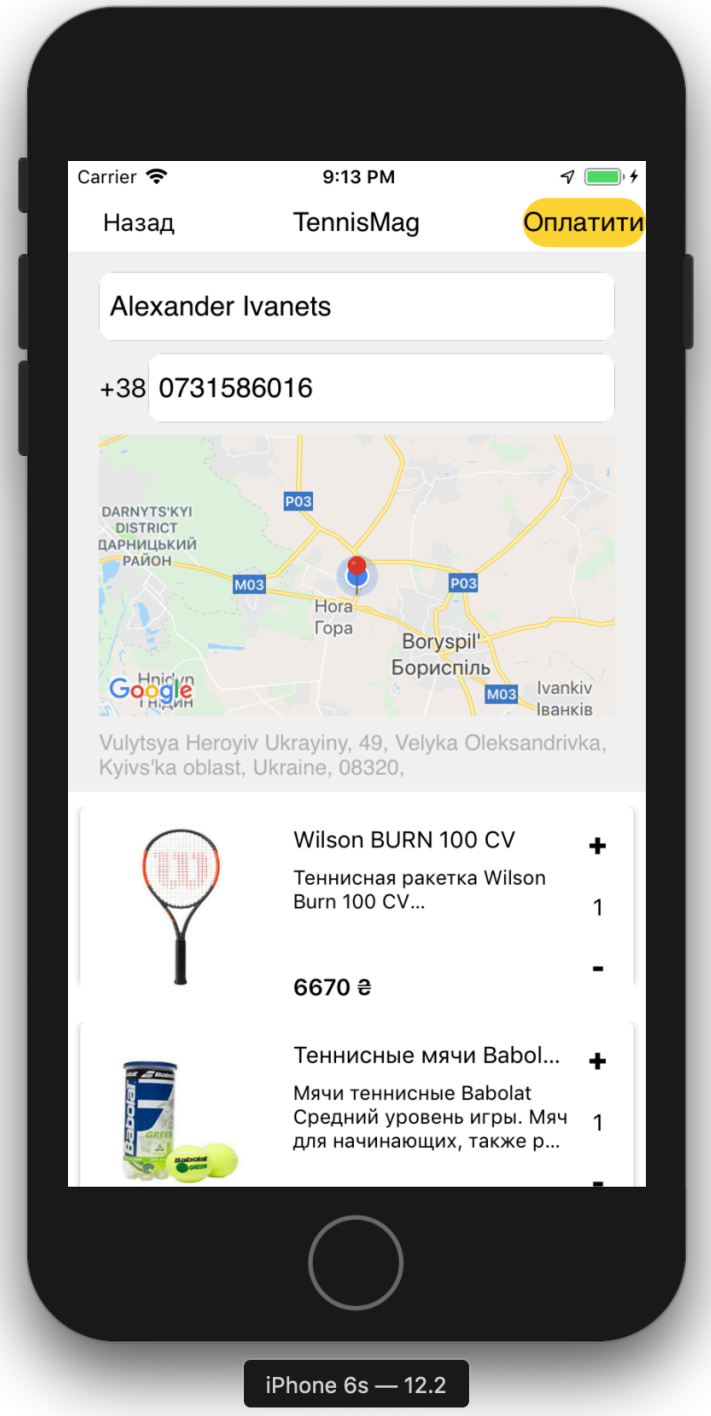


Рисунок 3.18 – Екран верифікації та вводу даних замовлення

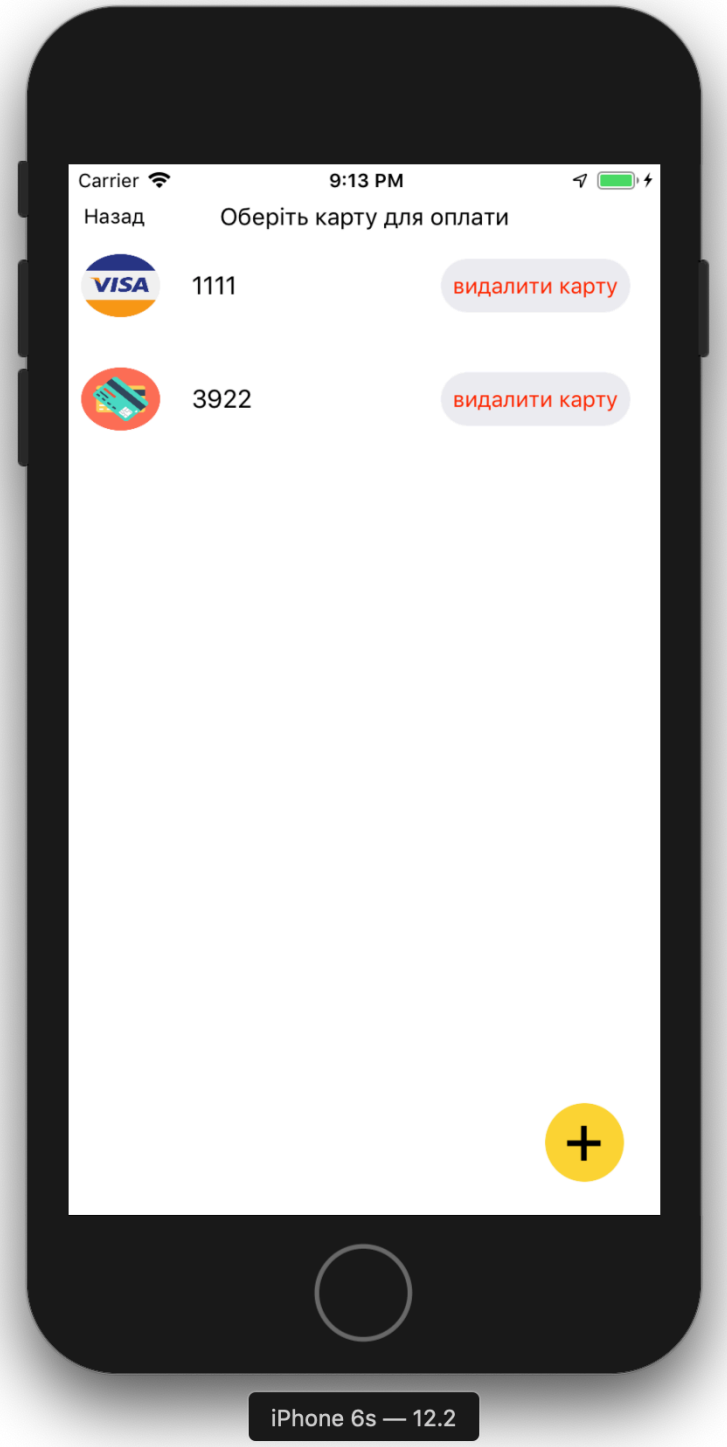


Рисунок 3.19 – Екран вибору карти для оплати замовлення

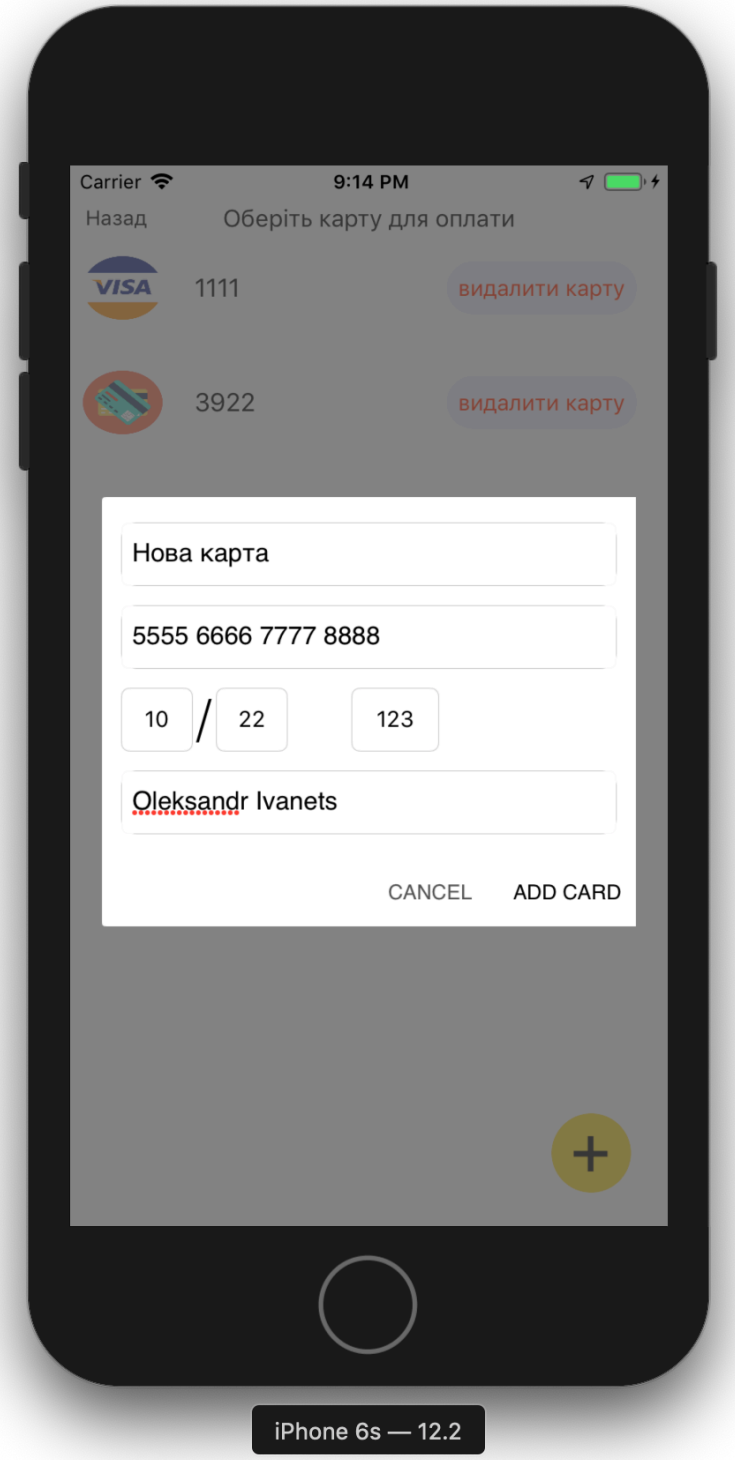


Рисунок 3.20 – Екран вибору карти для оплати з відкритим вікном додовання додаткової банківської карти

ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі було складено детальну інструкцію користувача для мобільного застосунку під платформу iOS. Кожен крок йде в тому самому порядку, що буде проходити користувач під час першого використання програмного продукту.

ВИСНОВКИ

Під час виконання дипломного проекту було проаналізовано ринок електронної комерції та різноманітні найвідоміші сервіси, котрі вже є на ринку.

Було проаналізовано найновітніші методології, паттерни та інструменти розробки під мобільну платформу iOS, та обрано список найоптимальніших технологій для розроблення програмного забезпечення.

Було розроблено список функціональних та нефункціональних вимог до розробленого застосунку.

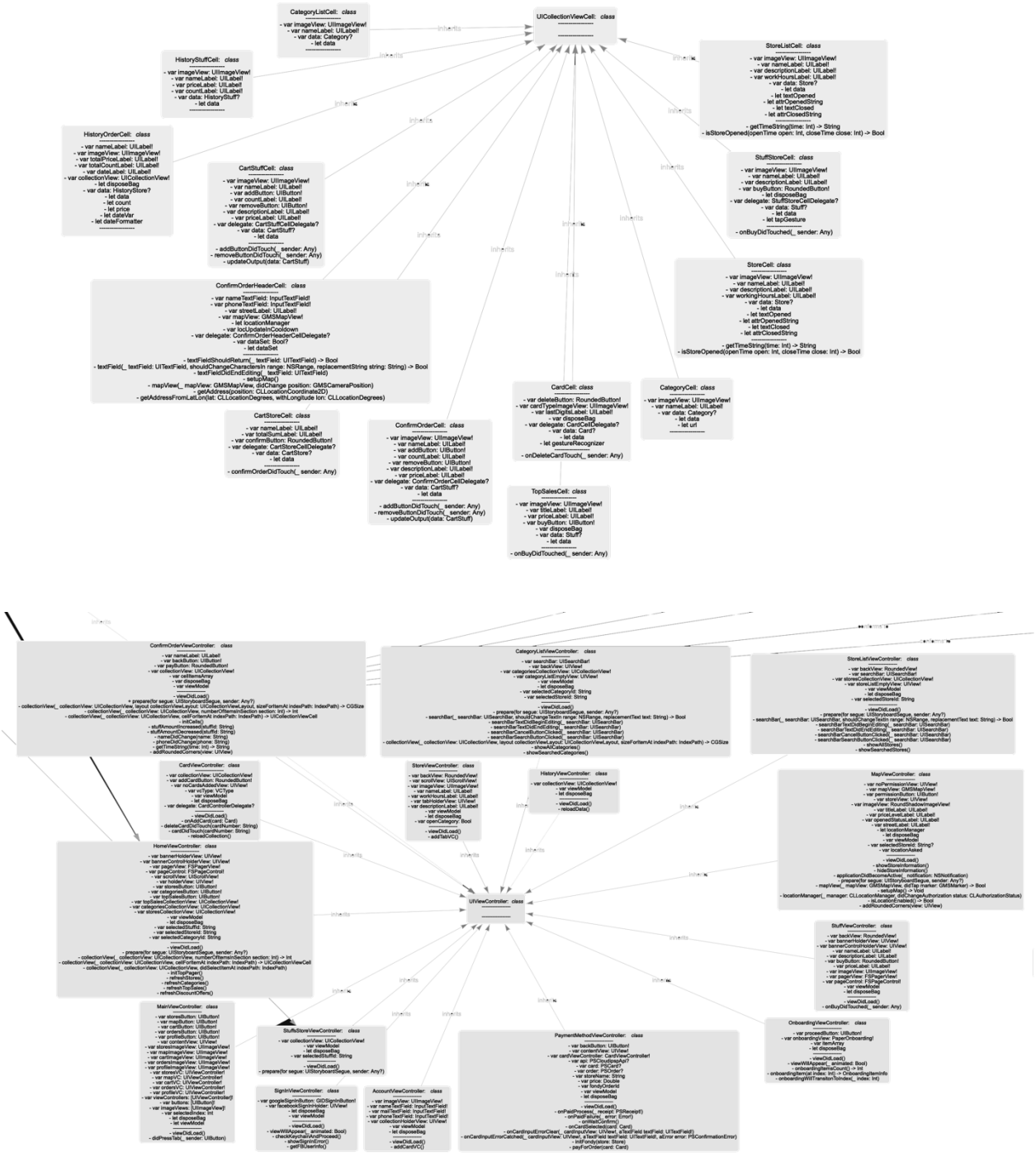
Було вивчено та проаналізовано питання безпеки використання застосунку, адже в ньому є критичні для безпеки модулі, такі як використання банківських карт задля оплати товарів.

Була розроблена розширена та поетапна інструкція користувача з поясненнями щодо кожного пункту.

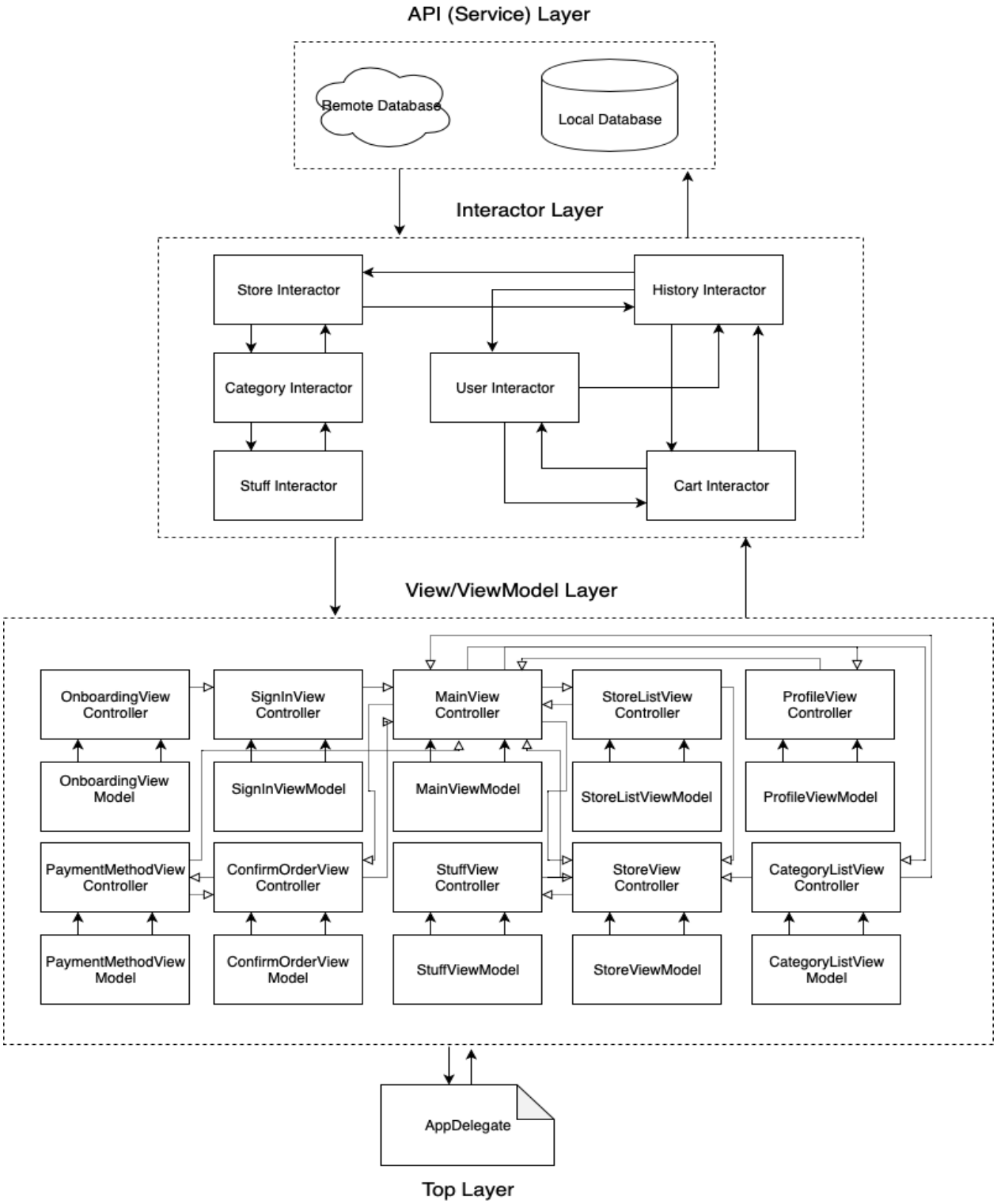
Розроблений мобільний застосунок повністю покрив всі описані функціональні та нефункціональні вимоги, котрі були поставлені перед ним. За допомогою нього автентифікований користувач може отримувати нотифікації про спеціальні пропозиції від магазинів поруч, переглядати список магазинів, категорій, переглядати інформацію про товар, додавати товари в кошик та придбати їх.

ПЕРЕЛІК ПОСИЛАНЬ

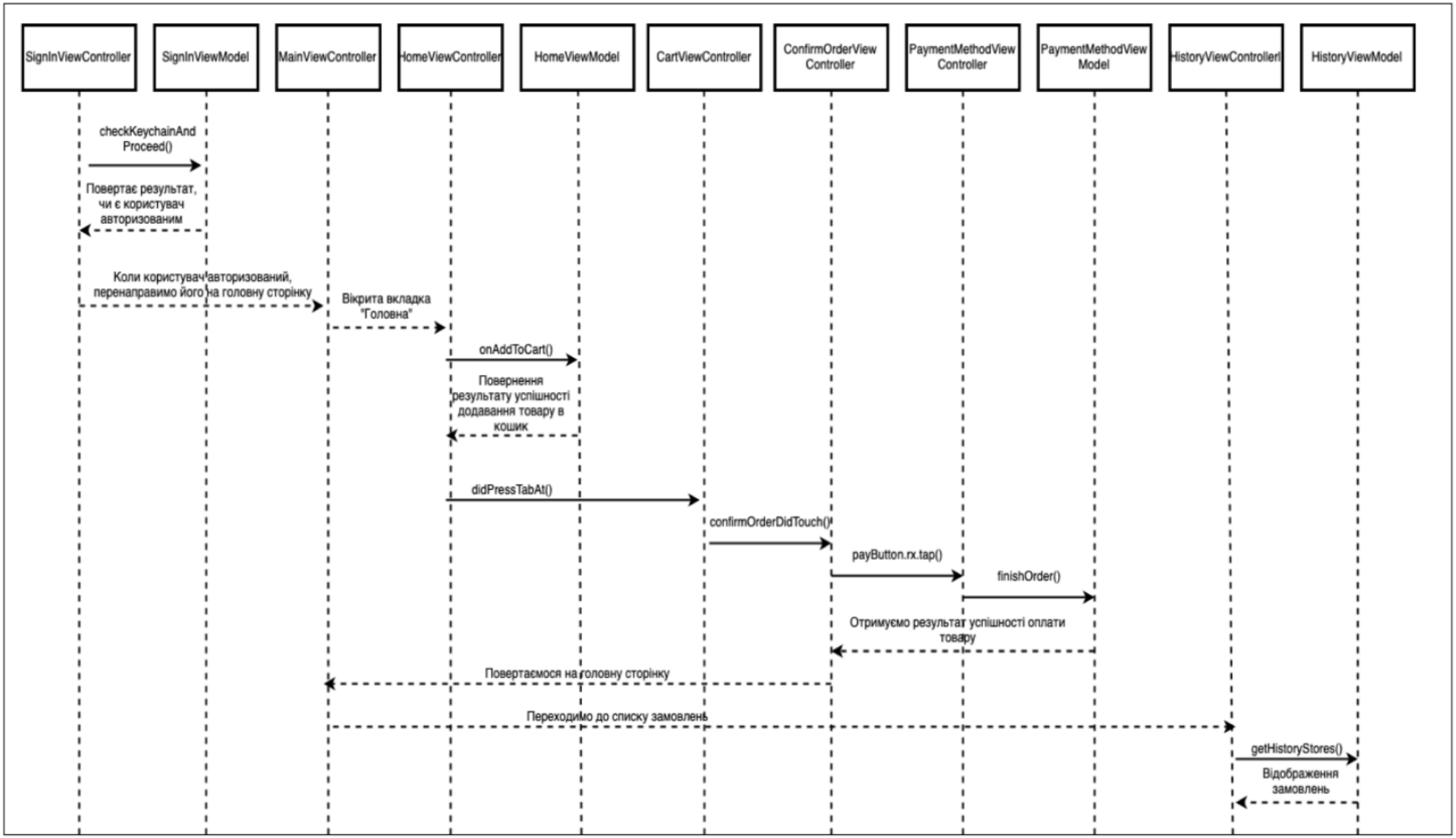
1. Какие бываюь интернет магазины [Електронний ресурс]: (Стаття) / Siteimage. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://www.siteimage.com.ua/blog/internet-magaziny/kakie-byvaiut-internet-magaziny. – Назва з екрана.
2. Создание приложения ToDo с помощью Realm и Swift[Електронний ресурс]: (Стаття) / Habr. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://habr.com/ru/post/272393/. – Назва з екрана.
3. Введение в Firebase: пишем простое социальное приложение на Swift [Електронний ресурс]: (Стаття) / Habr. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://habr.com/ru/post/277941/. – Назва з екрана.
4. Различие между MVVM и остальными MV*-паттернами [Електронний ресурс]: (Стаття) / Habr. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://habr.com/ru/company/mobileup/blog/313538/– Назва з екрана.



					ІАЛЦ.467200.004 Д1	
					Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS *Діаграма класів*	Літ. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Іванець О.А.				Аркуш 1 / Аркушів 1
Перевірив		Алещенко О.В.				
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ», ФІОТ, ІП-53
Затв.		Стіренко С.Г.				



					ІАЛЦ.467200.005 Д2	
					Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS *Діаграма компонентів*	*Літ.* / *Маса* / *Масштаб*
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Іванець О.А.*				*Аркуш 1* / *Аркушів 1*
Перевірив		*Алещенко О.В.*				
Н. Контр.		*Сімоненко В.П.*				*НТУУ «КПІ», ФІОТ, ІП-53*
Затв.		*Стіренко С.Г.*				



					ІАЛЦ.467200.006 ДЗ	
Зм.	Арк.	№ документа	Підпис	Дата	Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS	Літера / Маса / Масштаб
Розробив		Іванець О.А.				
Перевірив		Алещенко О.В.				Аркуш 1 / Аркушів 1
Н. кон.		Сімоненко В.П.			*Система електронної інвентаризації*	НТУУ "КПІ ім. І. Сікорського" кафедра ОТ, гр. ІП-53
Затвердив		Стіренко С.Г.				

ДОДАТОК 4

Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS

Лістинг програми

ІАЛЦ.467200.007 Д4

Аркушів 24

2019

```
class StoreInteractor {
    private let dbService: DbService
    private let apiService = ApiService()

    init(dbService: DbService) {
        self.dbService = dbService
    }

    func getStores() -> Maybe<Array<Store>> {
        return dbService.getStores()
    }

    func getStores(namePart name: String) -> Maybe<Array<Store>> {
        return dbService.getStores(namePart: name)
    }

    func getStore(id: String) -> Maybe<Store> {
        return dbService.getStore(id: id)
    }

    func getStore(categoryId id: String) -> Maybe<Store> {
        return dbService.getStore(categoryId: id)
    }

    func addStores(stores: Array<Store>) -> Maybe<Array<Store>> {
        return dbService.addStores(stores: stores)
    }


    func upsertStores() -> Observable<Bool> {
        return apiService.getStores().flatMap({stores in
            return self.dbService.addStores(stores: stores)}).map({stores in return true})
    }

}
```

```
class CategoryInteractor {

    private let dbService: DbService
    private let apiService = ApiService()

    init(dbService: DbService) {
        self.dbService = dbService
    }

    func getAllCategories() -> Maybe<Array<Category>> {
        return dbService.getAllCategories()
    }

    func getCategoriesByStore(storeId: String) -> Maybe<Array<Category>> {
        return dbService.getCategories(storeId: storeId)
    }

    func getCategoriesByName(namePart name: String) -> Maybe<Array<Category>> {
        return dbService.getCategories(namePart: name)
    }

    func addCategories(categories: Array<Category>) -> Maybe<Array<Category>> {
        return dbService.addCategories(categories: categories)
    }

    func upsertCategories() -> Observable<Bool> {
        return apiService.getCategories().flatMap({categories in return
self.dbService.addCategories(categories: categories)}).map{categories in return true}
    }

}

class StuffInteractor {
    private let dbService: DbService
```

```
private let apiService = ApiService()

init(dbService: DbService) {
   self.dbService = dbService
}

func addStuff(stuffArray: Array<Stuff>) -> Maybe<Array<Stuff>> {
   return dbService.addStuff(stuffArray: stuffArray)
}

func getStuff(stuffId: String) -> Maybe<Stuff> {
   return dbService.getStuff(stuffId: stuffId)
}

func getAllStuff() -> Maybe<Array<Stuff>> {
   return dbService.getAllStuff()
}

func getStuffByCategory(categoryId: String) -> Maybe<Array<Stuff>> {
   return dbService.getStuff(categoryId: categoryId)
}

func getAllDiscountOffers() -> Maybe<Array<DiscountOffer>> {
   return dbService.getAllDiscountOffers()
}

func upsertDiscountOffers() -> Observable<Bool> {
   return apiService.getDiscountOffers().flatMap({offers in
      return self.dbService.addDiscountOffer(array: offers)
   }).map({array in return true})
}

func upsertStuff() -> Observable<Bool> {
   return apiService.getStuffs().flatMap({stuffs in
      return self.dbService.addStuff(stuffArray: stuffs)}).map({array in return true})
```

```
    }
    }
    class CartInteractor {
        private let dbService: DbService

        init(dbService: DbService) {
            self.dbService = dbService
        }

        func incrementStuff(stuffId id: String) -> Maybe<Stuff> {
            return dbService.addStuffToOrder(stuffId: id)
        }

        func decrememntStuff(stuffId id: String) -> Maybe<Bool> {
            return dbService.decrementStuffFromOrder(stuffId: id)
        }

        func getStuffList() -> Maybe<Array<Stuff>> {
            return dbService.getStuffListFromOrder()
        }

        func deleteAllStoreStuffs(storeId id: String) -> Maybe<Bool> {
            return dbService.deleteAllStuffsRelatedToStore(storeId: id)
        }
    }

    class HistoryInteractor {
        private let dbService: DbService

        init(dbService: DbService) {
            self.dbService = dbService
        }

        func addHistoryStores(stores: Array<HistoryStore>) -> Maybe<Array<HistoryStore>>
{
```

```
        return dbService.addHistoryStores(stores: stores)
    }

    func getHistoryStores() ->Maybe<Array<HistoryStore>> {
        return dbService.getHistoryStores()
    }

    func addHistoryStore(store: HistoryStore) -> Maybe<HistoryStore> {
        return dbService.addHistoryStore(store: store)
    }

    func deletwHistoryStores() -> Maybe<Bool> {
        return dbService.deleteHistoryStores()
    }
}

class CardInteractor {
    let dbService: DbService

    init(dbService: DbService) {
        self.dbService = dbService
    }

    func addCard(card: Card) -> Maybe<Card> {
        return dbService.addCard(card: card)
    }

    func getCards() -> Maybe<Array<Card>> {
        return dbService.getCards()
    }

    func getCard(number: String) -> Maybe<Card> {
        return dbService.getCard(number: number)
    }
```

```
    func deleteCard(number: String) -> Maybe<Bool> {
        return dbService.deleteCard(number: number)
    }
}

class UserInteractor {
    let dbService: DbService

    init(dbService: DbService) {
        self.dbService = dbService
    }


    func updateUser(name: String) -> Maybe<Bool> {
        return dbService.updateUserInfo(updatedName: name)
    }


    func getUserInfo() -> Maybe<UserInfo> {
        return self.dbService.getUserInfo()
    }

    func addUserInfo(userInfo: UserInfo) -> Maybe<UserInfo> {
        return dbService.addUserInfo(userInfo: userInfo)
    }

}

class DbService {

    func addStores(stores: Array<Store>) -> Maybe<Array<Store>> {
        print("sasha: adding stores into db")
        let realm = try! Realm()
        try! realm.write {
            realm.delete(realm.objects(Store.self))
```

```
                for store in stores {
                    realm.add(store)
                    realm.objects(Store.self).filter({$0.id ==
store.id}).first?.categories.append(objectsIn: store.categoriesArray)
                }
            }
            return Maybe.just(stores)
        }

        func getStores() -> Maybe<Array<Store>> {
            let realm = try! Realm()
            let store = realm.objects(Store.self).toArray()
            return Maybe.just(store)
        }

        func getStores(namePart name: String) -> Maybe<Array<Store>> {
            let realm = try! Realm()
            let store = realm.objects(Store.self).filter("name contains
'\(name.lowercased())'").toArray()
            return Maybe.just(store)
        }

        func getStore(id: String) -> Maybe<Store> {
            let realm = try! Realm()
            let store = realm.objects(Store.self).filter({$0.id == id}).first

            if store == nil {

                return Maybe.empty()
            }

            return Maybe.just(store!)
        }

        func getStore(categoryId id: String) -> Maybe<Store> {
```

```
        let realm = try! Realm()
        let category = realm.objects(Category.self).filter({$0.categoryId == id}).first

        if(category == nil) {
            return Maybe.empty()
        }

        let store = realm.objects(Store.self).filter({$0.id == category!.storeId}).first

        if store == nil {

            return Maybe.empty()
        }

        return Maybe.just(store!)
    }

    func addStuff(stuffArray: Array<Stuff>) -> Maybe<Array<Stuff>> {
        let realm = try! Realm()
        try! realm.write {
            realm.delete(realm.objects(Stuff.self))
            for stuff in stuffArray {
                realm.add(stuff)
//                realm.objects(Stuff.self).filter({$0.stuffId ==
stuff.stuffId}).first?.categories.append(objectsIn: store.categoriesArray)
            }
        }
        return Maybe.just(stuffArray)
    }

    func getAllStuff() -> Maybe<Array<Stuff>> {
        let realm = try! Realm()
        let stuffList = realm.objects(Stuff.self).toArray()
        return Maybe.just(stuffList)
    }
```

```
    func getStuff(categoryId: String) -> Maybe<Array<Stuff>> {
        let realm = try! Realm()
        let stuffArray = realm.objects(Stuff.self).filter("storeCategoryId ==
'\(categoryId)'").toArray()
        return Maybe.just(stuffArray)
    }

    func getStuff(stuffId: String) -> Maybe<Stuff> {
        let realm = try! Realm()
        let stuff = realm.objects(Stuff.self).filter({$0.stuffId == stuffId}).first

        if stuff == nil {

            return Maybe.empty()
        }

        return Maybe.just(stuff!)
    }

    func addCategories(categories: Array<Category>) -> Maybe<Array<Category>> {
        let realm = try! Realm()
        try! realm.write {
            realm.delete(realm.objects(Category.self))
            for category in categories {
                realm.add(category)
            }
        }
        return Maybe.just(categories)
    }

    func getCategories(storeId: String) -> Maybe<Array<Category>> {
        let realm = try! Realm()
        let categories = realm.objects(Category.self).filter("storeId == '\(storeId)'").toArray()
        return Maybe.just(categories)
```

```
    }

    func getAllCategories() -> Maybe<Array<Category>> {
        let realm = try! Realm()
        let categories = realm.objects(Category.self).toArray()
        return Maybe.just(categories)
    }

    func getCategories(namePart name: String) -> Maybe<Array<Category>> {
        let realm = try! Realm()
        let categories = realm.objects(Category.self).filter("name contains
'\(name)'").toArray()
        return Maybe.just(categories)
    }

    func addStuffToOrder(stuffId id: String) -> Maybe<Stuff> {
        let realm = try! Realm()
        let stuff = realm.objects(Stuff.self).filter("stuffId == '\(id)'").first
        if stuff == nil {
            return Maybe.empty()
        }

        if (UserDefaults.standard.data(forKey: "orderStorageIdentifier") == nil) {
            let emptyOrder = Order()

            let encoder = JSONEncoder()
            if let encoded = try? encoder.encode(emptyOrder) {
                UserDefaults.standard.set(encoded, forKey: "orderStorageIdentifier")
                UserDefaults.standard.synchronize()
            }
        }

        let decoder = JSONDecoder()
        if let data = UserDefaults.standard.data(forKey: "orderStorageIdentifier"),
            let order = try? decoder.decode(Order.self, from: data) {
```

```
            order.orderItemsArray.append(stuff!)

            let encoder = JSONEncoder()
            if let encoded = try? encoder.encode(order) {
                UserDefaults.standard.set(encoded, forKey: "orderStorageIdentifier")
            }
            UserDefaults.standard.synchronize()
            return Maybe.just(stuff!)
        } else {
            return Maybe.empty()
        }
    }

    func decrementStuffFromOrder(stuffId id: String) -> Maybe<Bool> {
        let realm = try! Realm()
        let stuff = realm.objects(Stuff.self).filter("stuffId == '\(id)'").first
        if stuff == nil {
            return Maybe.empty()
        }

        if (UserDefaults.standard.data(forKey: "orderStorageIdentifier") == nil) {
            let emptyOrder = Order()

            let encoder = JSONEncoder()
            if let encoded = try? encoder.encode(emptyOrder) {
                UserDefaults.standard.set(encoded, forKey: "orderStorageIdentifier")
                UserDefaults.standard.synchronize()
            }
        }

        let decoder = JSONDecoder()
        if let data = UserDefaults.standard.data(forKey: "orderStorageIdentifier"),
            let order = try? decoder.decode(Order.self, from: data) {
            if(!order.orderItemsArray.contains(where: {$0.stuffId == stuff!.stuffId})) {
                return Maybe.just(true)
```

```
                } else {
                    var index: Int?
                    for (i, eStuff):(Int, Stuff) in order.orderItemsArray.enumerated() {
                        if(eStuff.stuffId == stuff!.stuffId) {
                            index = i
                        }
                    }
                    if(index != nil) {
                        order.orderItemsArray.remove(at: index!)
                    }
                }
                let encoder = JSONEncoder()
                if let encoded = try? encoder.encode(order) {
                    UserDefaults.standard.set(encoded, forKey: "orderStorageIdentifier")
                }
                UserDefaults.standard.synchronize()
                return Maybe.just(true)
            } else {
                return Maybe.empty()
            }
        }

        func deleteAllStuffsRelatedToStore(storeId id: String) -> Maybe<Bool> {
            let decoder = JSONDecoder()
            if let data = UserDefaults.standard.data(forKey: "orderStorageIdentifier"),
                let order = try? decoder.decode(Order.self, from: data) {
                let newArr = order.orderItemsArray.filter({
                    print("sasha: parentStoreId = \($0.parentStoreId)")
                    return $0.parentStoreId != id})
                print("sasha: storeId = \(id) prev count = \(order.orderItemsArray.count) newArr
count = \(newArr.count)")
                order.orderItemsArray = newArr
                let encoder = JSONEncoder()
                if let encoded = try? encoder.encode(order) {
                    print("sasha: will putit into db...")
```

```
                UserDefaults.standard.set(encoded, forKey: "orderStorageIdentifier")
                UserDefaults.standard.synchronize()
                return Maybe.just(true)
            }
            return Maybe.just(false)
        } else {
            return Maybe.empty()
        }
    }

    func getStuffListFromOrder() -> Maybe<Array<Stuff>> {
        let decoder = JSONDecoder()
        if let data = UserDefaults.standard.data(forKey: "orderStorageIdentifier"),
        let order = try? decoder.decode(Order.self, from: data) {
            return Maybe.just(order.orderItemsArray)
        } else {
            return Maybe.empty()
        }
    }

    func addHistoryStores(stores: Array<HistoryStore>) -> Maybe<Array<HistoryStore>>
{
        let realm = try! Realm()
        try! realm.write {
            realm.delete(realm.objects(HistoryStore.self))
            for store in stores {
                realm.add(store)
                realm.objects(HistoryStore.self).filter({$0.id ==
store.id}).first?.cartStuffs.append(objectsIn: store.cartStuffArray)
            }
        }
        return Maybe.just(stores)
    }

    func addHistoryStore(store: HistoryStore) -> Maybe<HistoryStore> {
```

```
        let realm = try! Realm()
        try! realm.write {
            realm.add(store)
            realm.objects(HistoryStore.self).filter({$0.id ==
store.id}).first?.cartStuffs.append(objectsIn: store.cartStuffArray)
        }
        return Maybe.just(store)
    }

    func getHistoryStores() ->Maybe<Array<HistoryStore>> {
        let realm = try! Realm()
        let store = realm.objects(HistoryStore.self).toArray()
        return Maybe.just(store)
    }

    func deleteHistoryStores() -> Maybe<Bool> {
        let realm = try! Realm()
        realm.delete(realm.objects(HistoryStore.self))
        return Maybe.just(true)
    }

    func addCard(card: Card) -> Maybe<Card> {
        let realm = try! Realm()
        try! realm.write {
            realm.add(card)
        }
        return Maybe.just(card)
    }

    func getCards() -> Maybe<Array<Card>> {
        let realm = try! Realm()
        let cards = realm.objects(Card.self).toArray()
        return Maybe.just(cards)
    }
```

```
func getCard(number: String) -> Maybe<Card> {
    let realm = try! Realm()
    let dish = realm.objects(Card.self).filter({$0.number == number}).first
    if(dish == nil) {
        return Maybe.empty()
    }
    return Maybe.just(dish!)
}

func deleteCard(number: String) -> Maybe<Bool> {
    let realm = try! Realm()
    let dish = realm.objects(Card.self).filter({$0.number == number}).first
    if(dish == nil) {
        return Maybe.just(false)
    }
    try! realm.write {
        realm.delete(realm.objects(Card.self).filter({$0.number == number}))
    }
    return Maybe.just(true)
}

func updateUserInfo(updatedName name: String) -> Maybe<Bool> {
    let realm = try! Realm()

    var userInfo: UserInfo?
    try! realm.write {
        realm.objects(UserInfo.self).first?.name = name
        userInfo = realm.objects(UserInfo.self).first
    }

    if(userInfo == nil) {
        return Maybe.just(false)
    }

    return Maybe.just(true)
```

```
}

func updateUserInfo(updatedPhone phone: String) -> Maybe<Bool> {
    let realm = try! Realm()

    var userInfo: UserInfo?
    try! realm.write {
        realm.objects(UserInfo.self).first?.phone = phone
        userInfo = realm.objects(UserInfo.self).first
    }

    if(userInfo == nil) {
        return Maybe.just(false)
    }

    return Maybe.just(true)
}

func addUserInfo(userInfo info: UserInfo) -> Maybe<UserInfo> {
    let realm = try! Realm()

    let userInfo = realm.objects(UserInfo.self).first

    if userInfo == nil {
        try! realm.write {
            realm.add(info)
        }
    } else {
        userInfo!.email = info.email
        userInfo!.phone = info.phone
        userInfo!.name = info.name
        userInfo!.id = info.id
        userInfo!.photoUrl = info.photoUrl
    }
```

```
            return Maybe.just(info)
        }

        func getUserInfo() -> Maybe<UserInfo> {
            let realm = try! Realm()

            let userInfo = realm.objects(UserInfo.self).first
            return Maybe.just(userInfo ?? UserInfo())

        }

        func getAllDiscountOffers() -> Maybe<Array<DiscountOffer>> {
            let realm = try! Realm()
            let stuffList = realm.objects(DiscountOffer.self).toArray()
            return Maybe.just(stuffList)
        }

        func addDiscountOffer(array: Array<DiscountOffer>) ->
Maybe<Array<DiscountOffer>> {
            let realm = try! Realm()
            try! realm.write {
                realm.delete(realm.objects(DiscountOffer.self))
                for item in array {
                    realm.add(item)
                }
            }
            return Maybe.just(array)
        }

    }

    class ApiService {
        var ref: DatabaseReference!

        init() {
```

```
        ref = Database.database().reference()
    }

    func getStores() -> Observable<Array<Store>> {
        return ref.child("stores").rx.observeEvent(.value).map({(snap: DataSnapshot) in
            var storeArray = Array<Store>()
            let childArray = snap.value as! NSArray
            for child in childArray {
                guard let dict = child as? [String:Any] else {
                    print("sasha: Error")
                    break
                }

                let store = Store(id: (dict["id"] as? String)!, merchantId: (dict["merchantId"] as? String)!, name: (dict["name"] as? String)!, photoUrl: (dict["photoUrl"] as? String)!, address: (dict["address"] as? String)!, descripton: (dict["description"] as? String)!, longitude: (dict["longitude"] as? Double)!, latitude: (dict["latitude"] as? Double)!, categories:/*todo*/ Array<StoreCategory>(), deleted: false, startWorkingSession: (dict["startWorkingSession"] as? Int)!, endWorkingSession: (dict["endWorkingSession"] as? Int)!)

                storeArray.append(store)
            }
            print("sasha: array size \(storeArray.count)")
            return storeArray
        })
    }

    func getCategories() -> Observable<Array<Category>> {
        return Observable.combineLatest(ref.child("stores").rx.observeEvent(.value), ref.child("categories").rx.observeEvent(.value)) { storesV, categV in
            var categoryArray = Array<Category>()
            let storeArr = storesV.value as! NSArray
            let catArr = categV.value as! NSArray
            for storeChild in storeArr {
                var storeId: String = ""
```

```
                var categoryId: String = ""
                var name: String = ""
                var photoUrl: String = ""
                var deleted: Bool = false


                guard let storeDict = storeChild as? [String:Any] else {
                    print("sasha: getCategories error!1")
                    break
                }

                let storeCatArr = storeDict["categories"] as! NSArray
                for storeCatChild in storeCatArr {
                    guard let storeCatDict = storeCatChild as? [String:Any] else {
                        print("sasha: getCategories error!2")
                        break
                    }
                    storeId = storeCatDict["storeId"] as? String ?? ""
                    categoryId = storeCatDict["parentCategoryId"] as? String ?? ""

                    for catChild in catArr {
                        guard let catDict = catChild as? [String:Any] else {
                            print("sasha: getCategories error!3")
                            break
                        }

                        print("sasha: categoryId \(categoryId) catDictId \(catDict["id"] as!
String)")
                        if((catDict["id"] as? String ?? "") == categoryId) {
                            name = catDict["name"] as? String ?? ""
                            photoUrl = catDict["photoUrl"] as? String ?? ""
                            deleted = catDict["deleted"] as? Bool ?? false

                        }
                    }
```

```
                    if(!categoryArray.contains(where: {$0.categoryId == categoryId}) &&
categoryId != "") {
                        let category = Category(storeId: storeId, categoryId: categoryId, name:
name, photoUrl: photoUrl, deleted: deleted)
                        categoryArray.append(category)
                    }
                    storeId = ""
                    categoryId = ""
                    name = ""
                    photoUrl = ""
                    deleted = false
                }
            }
            return categoryArray
        }

    }
    func getStuffs() -> Observable<Array<Stuff>> {

        return Observable.combineLatest(ref.child("stores").rx.observeEvent(.value),
ref.child("categories").rx.observeEvent(.value)) { storesV, categV in

            var stuffArray = Array<Stuff>()

            let storeArr = storesV.value as! NSArray

            let catArr = categV.value as! NSArray

            for storeChild in storeArr {


                guard let storeDict = storeChild as? [String:Any] else {

                    print("sasha: getStuffs error!1")

                    break

                }
```

```
                    let storeStuffArr = storeDict["stuffs"] as! NSArray
                    for storeStuffChild in storeStuffArr {
                        guard let storeStuffDict = storeStuffChild as? [String:Any] else {
                            print("sasha: getStuffs error!2")
                            break
                        }

                        //let stuffCatArr = storeStuffDict["stuffCategory"] as! NSArray

                        guard let stuffCatDict = storeStuffDict["stuffCategory"] as? [String:Any] else {
                            print("sasha: getStuffs error!3")
                            break
                        }
                        let stuffCatId = stuffCatDict["categoryId"] as? String

                        for catChild in catArr {
                            guard let catDict = catChild as? [String:Any] else {
                                print("sasha: getStuffs error!4")
                                break
                            }

                            if(catDict["id"] as? String ==  stuffCatId) {
                                let stuff = Stuff(stuffId: storeStuffDict["stuffId"] as? String ?? "",
storeCategoryId: stuffCatId ?? "", stuffCategoryId: stuffCatId ?? "", parentStoreId:
storeStuffDict["parentStoreId"] as? String ?? "", name: storeStuffDict["name"] as? String ?? "",
```

```
photoUrl: storeStuffDict["photoUrl"] as? String ?? "", rDescription: storeStuffDict["description"] as? String ?? "", price: storeStuffDict["price"] as? Int ?? 0, blocked: storeStuffDict["blocked"] as? Bool ?? false, deleted: storeStuffDict["deleted"] as? Bool ?? false)

                    dump(stuff)
                    stuffArray.append(stuff)
                }
            }

        }
        }
        return stuffArray
    }
}

func getDiscountOffers() -> Observable<Array<DiscountOffer>> {
    return ref.child("discountOffers").rx.observeEvent(.value).map({(snap: DataSnapshot) in
        var discountArray = Array<DiscountOffer>()
        let childArray = snap.value as! NSArray
        for child in childArray {
            guard let dict = child as? [String:Any] else {
                print("sasha: Error")
                break
            }
```

```
                    let discountOffer = DiscountOffer(id: dict["id"] as? String ?? "", stuffId:
dict["stuffId"] as? String ?? "", photoUrl: dict["photoUrl"] as? String ?? "")


                    discountArray.append(discountOffer)
                }
                return discountArray
            })
        }

}
```

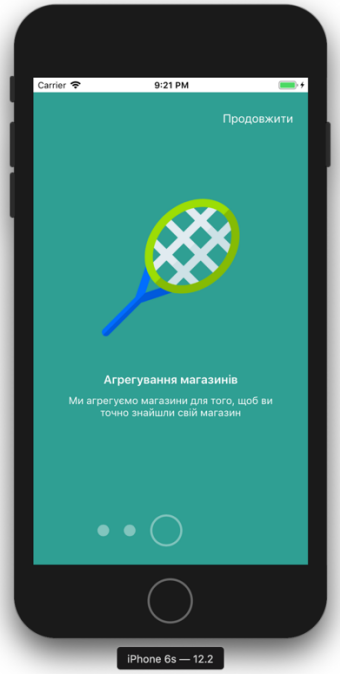
ДОДАТОК 5

Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS

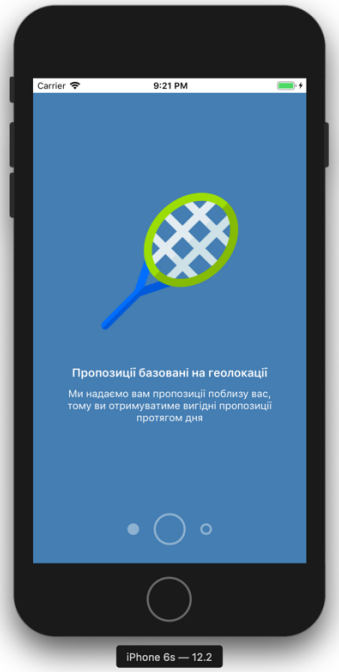
Скриншоти програми

ІАЛЦ.467200.008 Д5

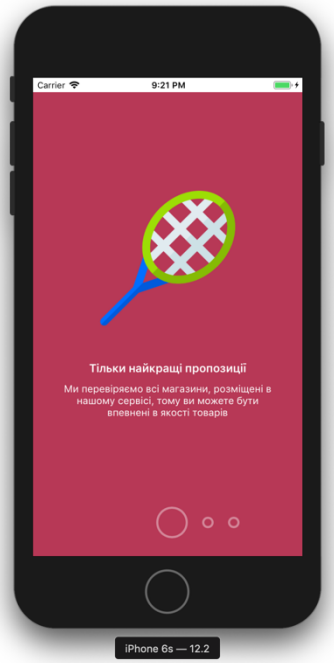
2019



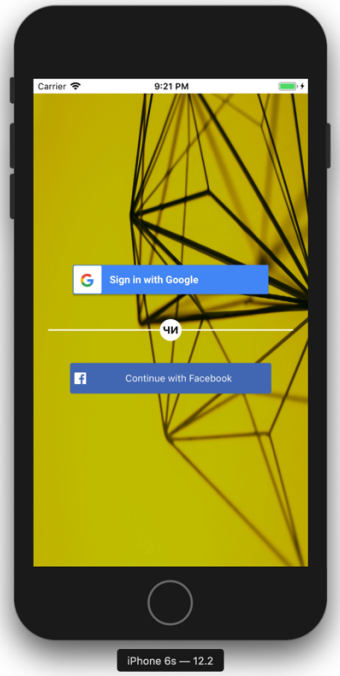
iPhone 6s — 12.2



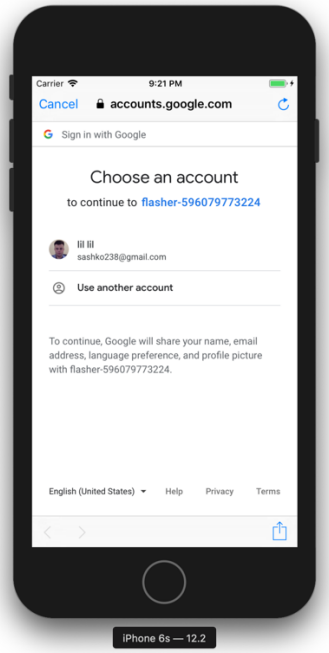
iPhone 6s — 12.2



iPhone 6s — 12.2

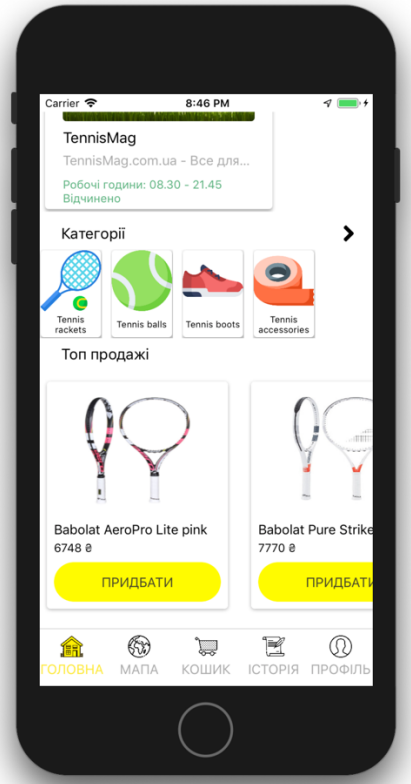


iPhone 6s — 12.2

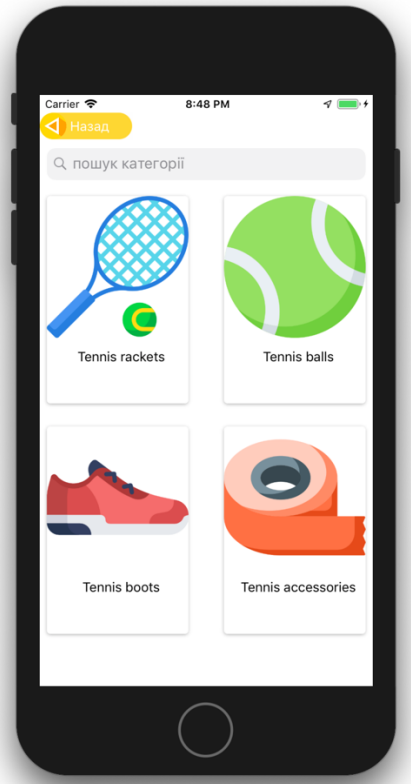


iPhone 6s — 12.2

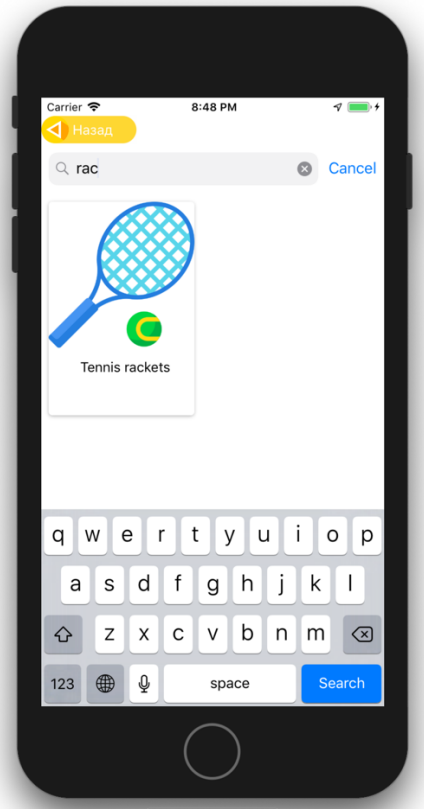
iPhone 6s — 12.2



iPhone 6s — 12.2

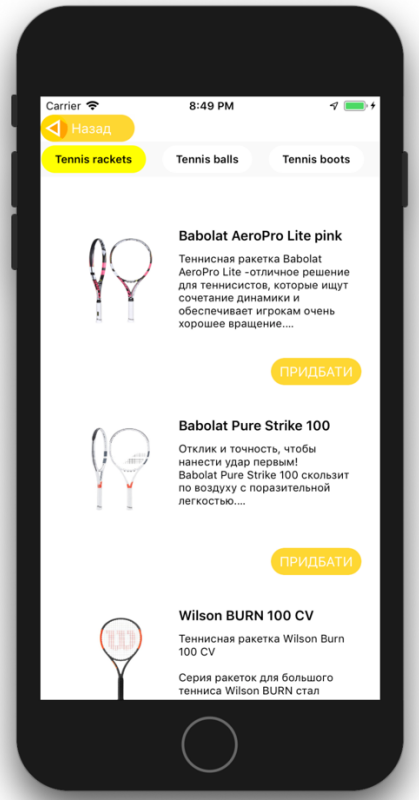


iPhone 6s — 12.2



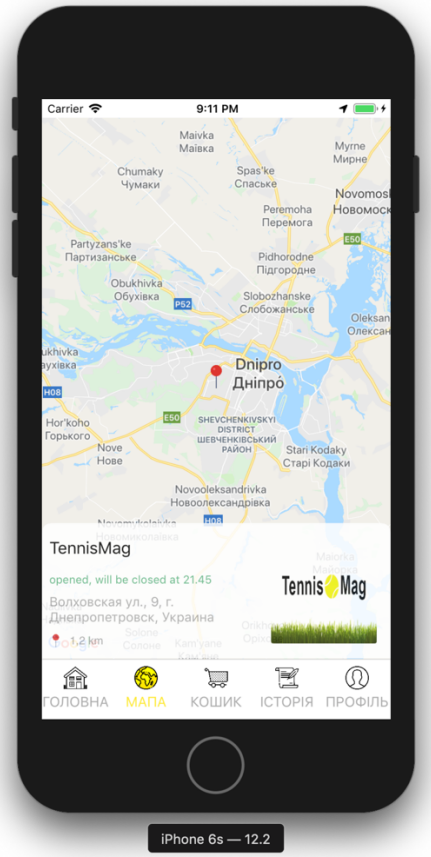
iPhone 6s — 12.2

iPhone 6s — 12.2

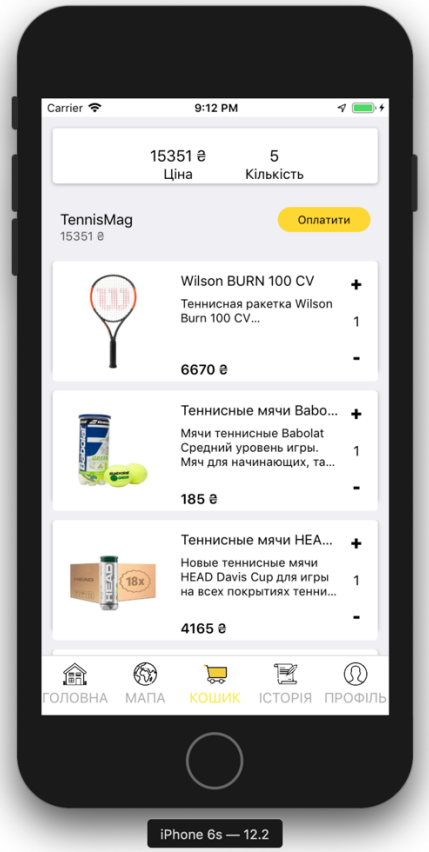


iPhone 6s — 12.2

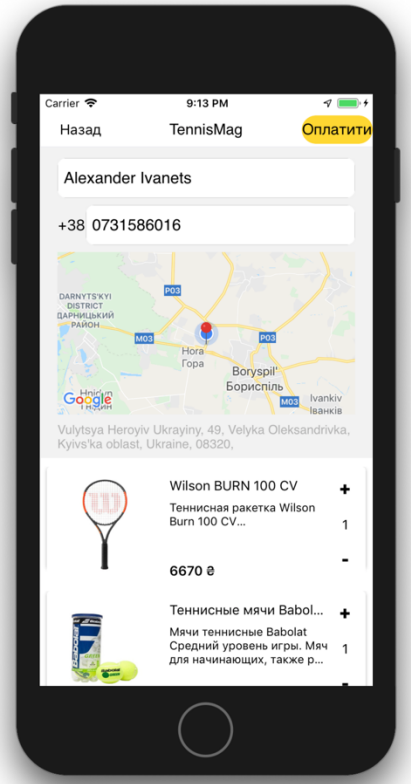
iPhone 6s — 12.2



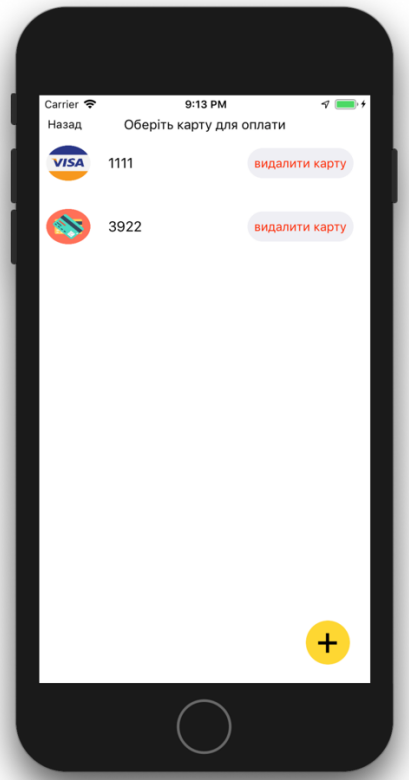
iPhone 6s — 12.2



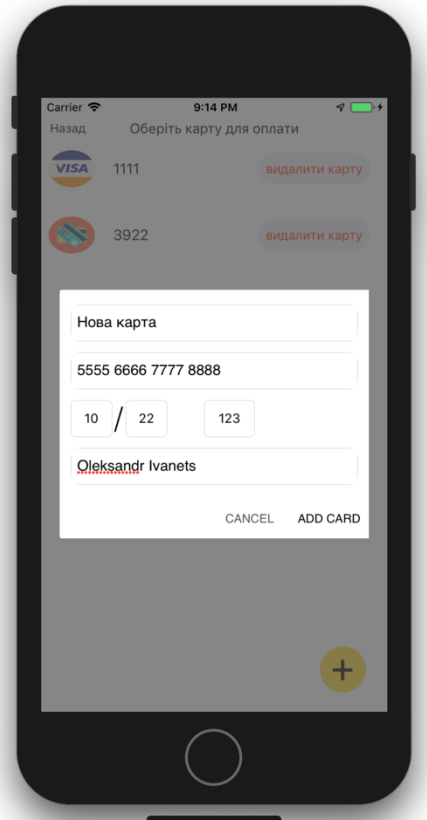
iPhone 6s — 12.2



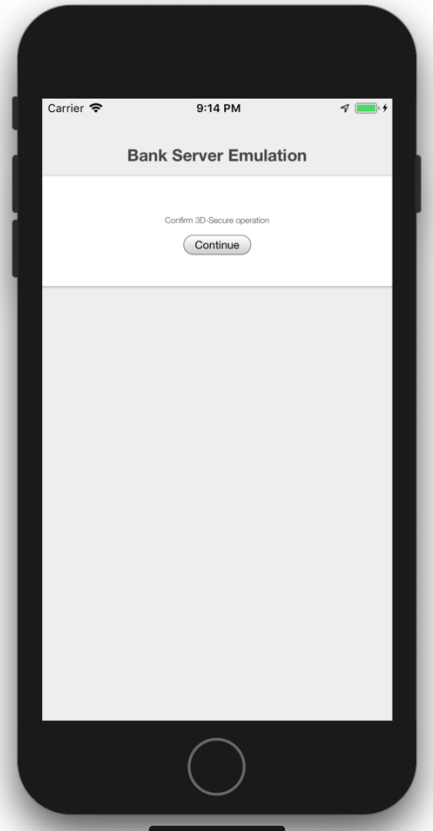
iPhone 6s — 12.2



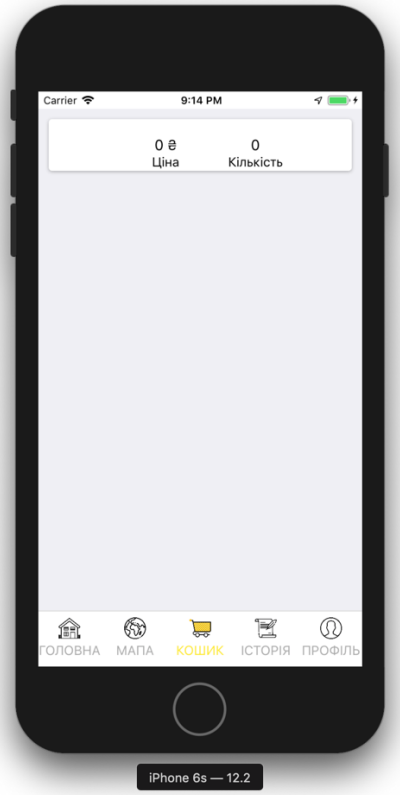
iPhone 6s — 12.2



iPhone 6s — 12.2



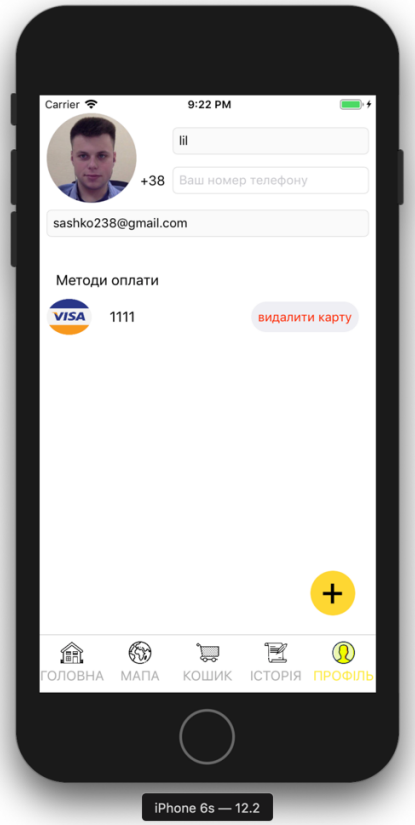
iPhone 6s — 12.2



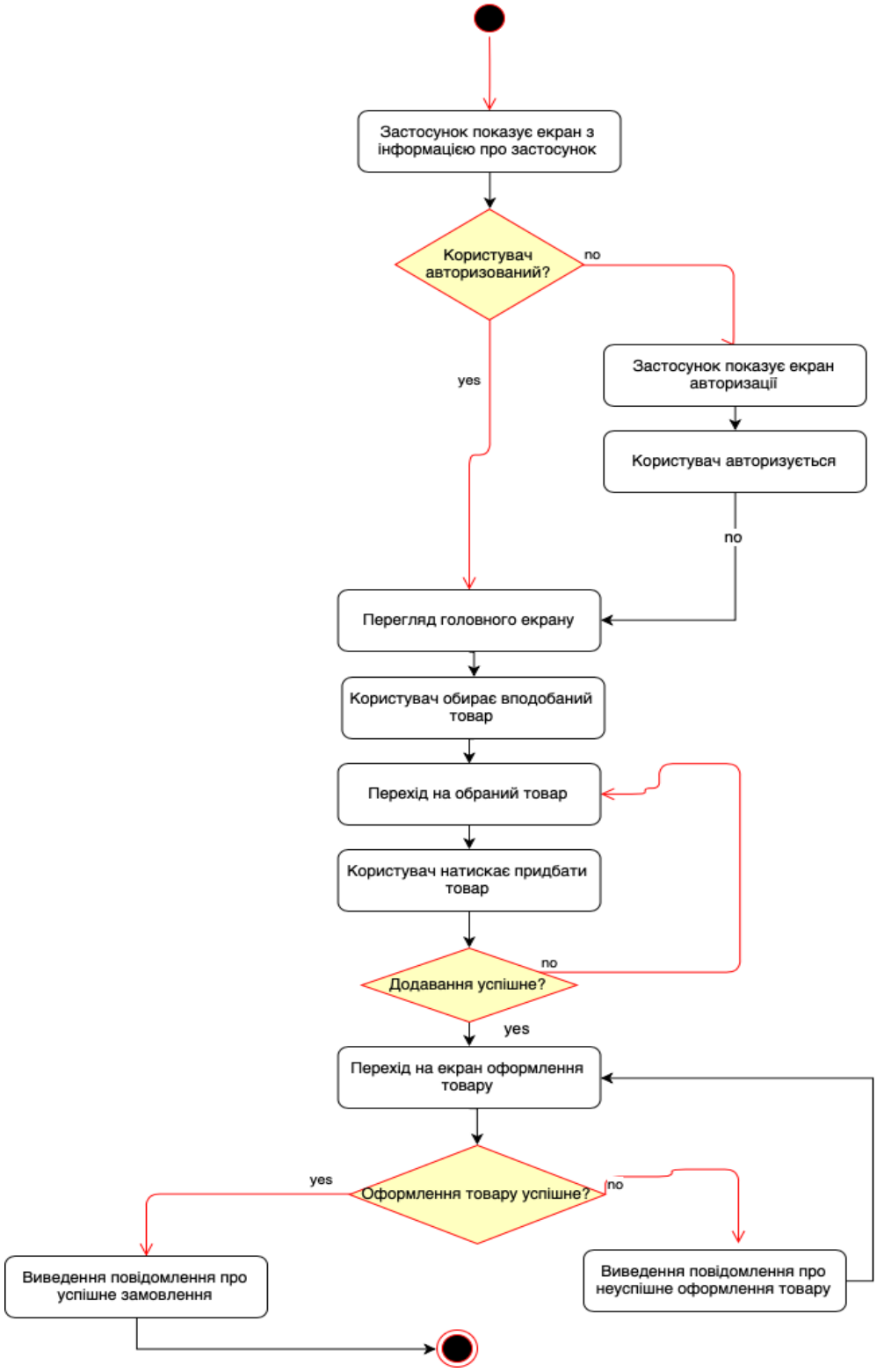
iPhone 6s — 12.2

iPhone 6s — 12.2

iPhone 6s — 12.2



iPhone 6s — 12.2



					ІАЛЦ.467200.009 Д6	
Зм.	Арк.	№ докум.	Підпис	Дата	Система придбання товарів з урахуванням місцезнаходження користувача для мобільної платформи iOS	Літ. / Маса / Масштаб
Розробив		Іванець О.А.			Алгоритм програми	
Перевірив		Алещенко О.В.				Аркуш 1 / Аркушів 1
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ», ФІОТ, ІП-53
Затв.		Стіренко С.Г.				